VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroje pro sjednocení datových zdrojů projektu Gloffer

Tools for unification of data sources project Gloffer

2018 Bc. Jakub Malchárek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Malchárek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroje pro sjednocení datových zdrojů projektu Gloffer
Tools for Unification of Data Sources Project Gloffer

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je provést analýzu, návrh a implementaci nástroje pro sjednocení datových zdrojů projektu Gloffer, jejich kombinace a unifikace do jednotného rozhraní, které bude sloužit jako vstup pro veřejné Gloffer API.

1. Student analyzuje dostupné technologie se zaměřením na platformu Linux a implementační prostředí a technologie MySQL, Elastic Search (Lucene, Solr), Apache, Redis, Rabbit MQ, Aerospike, PHP7 a Framework Nette. Pro účely implementace API zvolí vhodné technologie (PHP, JAVA atd.).
2. Student se zaměří na unifikaci komunikačního rozhraní pro fultextové vyhledávání (Lucene, Elastic Search, Solr), SQL a Non-SQL databáze (MongoDB, HBase, Casandra, Google Big table) a budování globálního komunikačního rozhraní pro API.
3. Dále student nastuduje a pro opakující se dotazy využije cachovací technologie Redis, Memcached a Aerospike.
4. Výsledkem bude vybudování globálního datového rozhraní, které bude konsolidovat různé datové zdroje, provádět cachování opakujících se dotazů a výsledků vyhledávání, generovat výstup v unifikovaném formátu bez ohledu na použitý vstupní zdroj dat. Unifikovaný datový zdroj bude sloužit jako základ pro Rest API projektu Gloffer.
5. Dále bude provedeno zátěžové testování, které určí limity a požadavky na technické vybavení vzhledem k rostoucímu počtu indexovaných položek (v řádech miliónů záznamů) a množství opakujících se a unikátních dotazů.
6. V závěru student provede srovnání výsledné implementace s existujícími řešeními a zhodnotí, zda je efektivní budovat vlastní informační platformu nebo využít již existující řešení a komponenty.

Seznam doporučené odborné literatury:

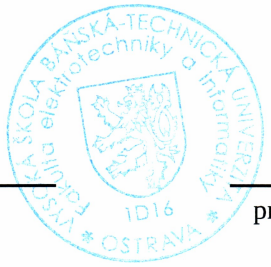
[1] GORMLEY, Clinton a Zachary TONG. Elasticsearch: the definitive guide. ISBN 1449358543.
[2] SHKLAR, Leon. a Rich. ROSEN. Web application architecture: principles, protocols and practices. 2nd ed. Hoboken, NJ: Wiley, c2009. ISBN 047051860x.
[3] SHIVAKUMAR, Shailesh Kumar. Architecting high performing, scalable and available enterprise web applications. ISBN 9780128022580.
[4] WEERAWARANA, Sanjiva. Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more. Upper Saddle River, NJ: Prentice Hall PTR, c2005. ISBN 0131488740.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radoslav Fasuga, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 19. dubna 2018 .....

Rád bych poděkoval panu Ing. Radoslavu Fasugovi, Ph.D. za odbornou pomoc a konzultaci při zpracování této diplomové práce a cenné rady v průběhu implementace.

Abstrakt

V této diplomové práci se zabývám analýzou dostupných technologií pro implementaci webového portálu Gloffer. Jsou zde popsány databáze (MySQL, Redis, MongoDB, Aerospike, Apache HBase, Apache Cassandra, Google Bigtable, Memcached), vyhledávače (Solr, Lucene, Elastic Search), webové servery (Apache HTTP server, Apache Tomcat), zprostředkovatelé zpráv (Rabbit MQ), distribuované výpočetní technologie (Apache Hadoop) a vývojové technologie (PHP 7, Nette Framework, Java, Spring Framework). Cílem je nejen popis těchto technologií, ale také návrh a implementace rozhraní pro sjednocení datových zdrojů projektu Gloffer v programovacím jazyce Java s využitím Spring Frameworku. Výstupem práce je inteligentní nástroj zpřístupňující data z více datových zdrojů. Závěr práce obsahuje výkonové testování vyvinutého nástroje.

Klíčová slova: Aerospike, Apache Cassandra, Apache Hadoop, Apache HBase, Apache HTTP server, Apache Tomcat, aplikační rozhraní, datové zdroje, Elastic Search, fulltext, Google Bigtable, index, Java, Lucene, Memcached, MongoDB, MySQL, Nette Framework, PHP, Rabbit MQ, Redis, REST, Solr, Spring Framework

Abstract

In this diploma thesis I deal with analysis of the available technologies for implementation of the Gloffer web portal. There are described databases (MySQL, Redis, MongoDB, Aerospike, Apache HBase, Apache Cassandra, Google Bigtable, Memcached), search engines (Solr, Lucene, Elastic Search), web servers (Apache HTTP server, Apache Tomcat), message brokers (Rabbit MQ), distributed computing technologies (Apache Hadoop) and develop technologies (PHP 7, Nette Framework, Java, Spring Framework). The target is not only description of this technologies but also a design and implementation of interface to unify data sources of Gloffer project in the Java programming language using the Spring Framework. Output of the thesis is an intelligent tool for accessing data from multiple data sources. The conclusion of the thesis includes performance testing of the developed tool.

Key Words: Aerospike, Apache Cassandra, Apache Hadoop, Apache HBase, Apache HTTP server, Apache Tomcat, application interface, data sources, Elastic Search, fulltext, Google Bigtable, index, Java, Lucene, Memcached, MongoDB, MySQL, Nette Framework, PHP, Rabbit MQ, Redis, REST, Solr, Spring Framework

Obsah

Seznam použitých zkratek a symbolů

ACID	–	Atomicity, Consistency, Isolation, Durability
API	–	Application Programming Interface
BDB	–	Berkeley Database
BM	–	Best Matching
BSON	–	Binary JavaScript Object Notation
CQL	–	Cassandra Query Language
CSRF	–	Cross-Site Request Forgery
CSV	–	Comma-Separated Values
DB	–	Database
DDL	–	Data Definition Language
DFI	–	Divergence From Independence
DFR	–	Divergence From Randomness
DML	–	Data Manipulation Language
EAN	–	European Article Number
GB	–	Gigabyte
GDPR	–	General Data Protection Regulation
GPL	–	General Public License
HHVM	–	HipHop Virtual Machine
HTML	–	HyperText Markup Language
HTTP	–	Hypertext Transfer Protocol
IB	–	Information Based
IDF	–	Inverse Document Frequency
IIS	–	Internet Information Services
JDBC	–	Java Database Connectivity
JDK	–	Java Development Kit
JIT	–	Just In Time
JRE	–	Java Runtime Environment
JSON	–	JavaScript Object Notation
JSP	–	JavaServer Pages
JVM	–	Java Virtual Machine
J2EE	–	Java 2 Enterprise Edition
LFU	–	Least Frequently Used
LM	–	Language Model
LRU	–	Least Recently Used
MPM	–	Multi-Processing Modules
NDB	–	Nucleic Acid Database

ODBC	–	Open Database Connectivity
PB	–	Petabyte
PHP	–	Hypertext Preprocessor
RAM	–	Random Access Memory
RDB	–	Relational Database
REST	–	Representational State Transfer
SOAP	–	Simple Object Access Protocol
SPOF	–	Single Point Of Failure
SQL	–	Structured Query Language
SSD	–	Solid-State Drive
SSL	–	Secure Sockets Layer
SSTable	–	Sorted Strings Table
TF	–	Term Frequency
TLS	–	Transport Layer Security
TTL	–	Time To Live
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
VPN	–	Virtual Private Network
XML	–	eXtensible Markup Language
XSS	–	Cross-Site Scripting
YAML	–	YAML Ain't Markup Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Webové portály jsou „bránou do světa internetu“. Poskytují jednotným způsobem informace z více různých zdrojů. Mají vysoké nároky na výkon a využívají spoustu technologií pro zpracování dotazů. Portál Gloffer není výjimkou ve využívání různých technologií, a proto si vývojáři tohoto portálu kladou za cíl sjednotit práci a způsob dotazování nad datovými zdroji portálu. Sjednocením těchto zdrojů se zjednoduší práce s daty při napojení na více koncových aplikací (mobilní, webová, případně desktopová).

Hlavním cílem mé práce je analýza dostupných technologií, dále pak návrh a implementace nástroje pro sjednocení datových zdrojů projektu Gloffer, který využívá pro ukládání různé typy databází. Pro persistentní ukládání dat využívá relační databázi MySQL a NoSQL databázi MongoDB, pro fulltextové vyhledávání slouží Elastic Search a cachované data se ukládají do Redisu. Cílem je tedy navrhnout nástroj, který přistupuje k těmto datovým zdrojům jednotným způsobem a tím programátora vyšších vrstev odstíní od nutnosti psaní specializovaných dotazů nad jednotlivými databázemi. Pro potřeby projektu portálu Gloffer je potřeba především rychlého vyhledávání, tudíž je kladen důraz na rychlost vykonání dotazů.

První kapitola je věnována popisu technologií využívaných pro vývoj webových portálů. Jsou zde popsány databáze (MySQL, Redis, MongoDB, Aerospike, Apache HBase, Apache Cassandra, Google Bigtable, Memcached), vyhledávače (Solr, Lucene, Elastic Search), webové servery (Apache HTTP server, Apache Tomcat), zprostředkovatelé zpráv (Rabbit MQ), distribuované výpočetní technologie (Apache Hadoop) a vývojové technologie (PHP 7, Nette Framework, Java, Spring Framework). Další část tvoří návrh samotného nástroje pro sjednocení datových zdrojů. Poté se věnuji implementaci nástroje ve vhodném programovacím jazyce a frameworku. Poslední část tvoří testování vykonání dotazů. Součástí je zhodnocení, zda se vyplatí vytvářet vlastní řešení nebo využít již existujících řešení třetích stran.

2 Portál Gloffer

Webový portál Gloffer je nová platforma zabývající se online nakupováním a prodejem. Vývoj začal roku 2015 a ke konci roku 2017 došlo ke spuštění Beta verze. Na vývoji se podílí malá skupinka studentů Fakulty elektrotechniky a informatiky VŠB-TUO pod vedením zkušeného pedagoga Ing. Radoslava Fasugy, Ph.D.

Hlavním cílem Glofferu je usnadnění vyhledání a zakoupení produktů formou chytrých poptávek. Zákazník vytvoří poptávku na produkt a obchodníci se snaží uspokojit jeho potřeby. Je to rozdílný způsob online nakupování, než na který jsme zvyklí ze srovnávačů cen jako je Heureka. Gloffer je vícejazyčný a multiměnový, využívá tedy překladů a přepočtů měn podle aktuálních kurzů.

Portál Gloffer je náročný na výkon, bezpečnost a množství dat a tudíž je potřeba výkonných serverů a technologií pro rychlé vyhledávání, indexování dat, autentizaci a autorizaci uživatelů. Důležitá je optimalizace dotazů a cachování výsledků pro pozdější opětovné vyhledávání. Použité technologie:

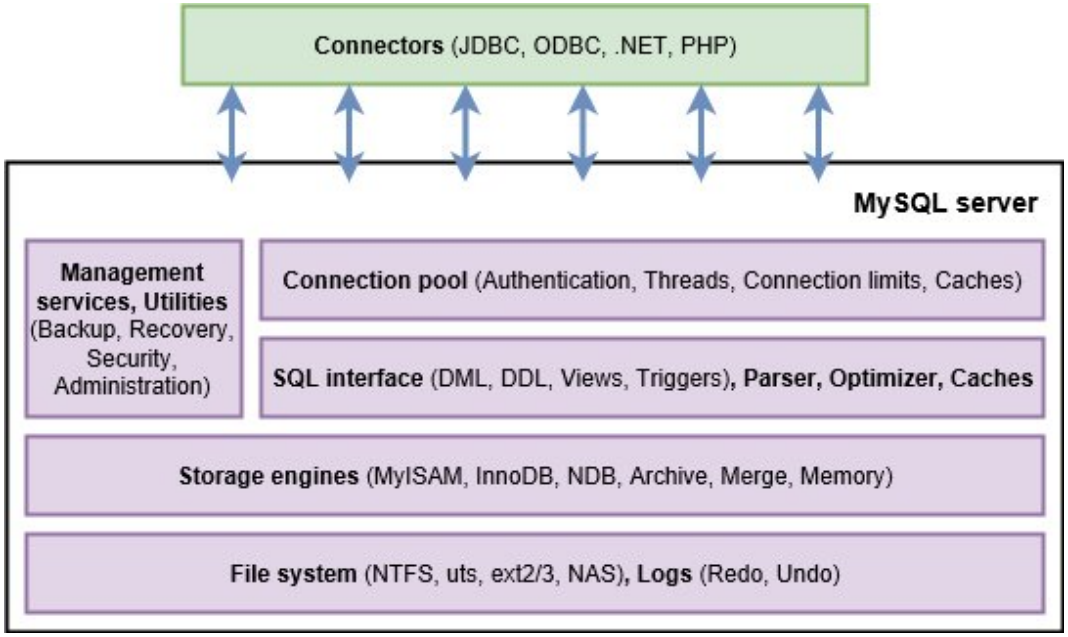
- Servery s operačním systémem Linux
- Datové sklady
  - MySQL - perzistentní data a cachované data
  - MongoDB - uložené produkty z „feedu“
  - Redis - cachované data
  - Elastic Search - fulltextové vyhledávání produktů
- Programovací jazyky (frameworky)
  - PHP (Nette Framework) - stávající webová aplikace
  - JAVA (Spring Framework) - REST API pro mobilní aplikaci, později napojení webové a administrátorské aplikace

3 Analýza technologií

3.1 MySQL

MySQL je relační databázový systém komunikující prostřednictvím jazyka SQL, který byl vyvinut v 90. letech 20. století. [5] Jedná se o jednoduše implementovatelný databázový systém, který se používá velice často v kombinaci s programovacím jazyce PHP za účelem vytváření webových aplikací. Výhody tohoto databázového systému jsou především jeho rychlost a pořizovací náklady (bezplatná licence GPL nebo komerční placená licence).

Architektura MySQL se odlišuje od architektur jiných databázových systémů. Na nejvyšší vrstvě se nachází nástroje potřebné ke komunikaci klienta se serverem. Druhá vrstva obsahuje kód pro parsování, analýzu, optimalizaci a všechny zabudované funkce. Na třetí vrstvě jsou datová úložiště. Ty se starají o ukládání a získávání všech dat uložených v MySQL. Datová úložiště poskytují své API, prostřednictvím kterého s nimi databázový server komunikuje. Pod touto vrstvou se nachází nízko úrovňové funkce, které se starají o transakce, získání řádku s určitým primárním klíčem atd. Architekturu MySQL serveru lze vidět na obrázku 1.



Obrázek 1: Architektura MySQL

MySQL podporuje několik datových úložišť - MyISAM, InnoDB, Memory, CSV, Archive, BDB. Datové úložiště je možné nastavit pro každou tabulku. Nejpoužívanějším úložištěm jsou MyISAM a InnoDB. MyISAM je rychlé datové úložiště, který však neumožňuje transakce a referenční integritu (cizí klíče). Jeho výhodou je, že umožňuje fulltextové vyhledávání a to díky textovým indexům. InnoDB je datové úložiště umožňující transakce a zajišťující referenční integritu. [5]

Nové verze MySQL podporují ukládání a práci s JSON soubory a prostorovými daty. S JSON soubory se v projektu Gloffer pracuje běžně, je tedy této přirozené podpory ze strany databázového serveru využíváno.

3.2 Lucene

Lucene je výkonná knihovna s otevřeným zdrojovým kódem pro fulltextové vyhledávání implementovaná v programovacím jazyce Java. První vydání Lucene proběhlo v roce 1999. Tato knihovna je použitelná v mnoha programovacích jazycích, jako je Object Pascal (LuceneKit), Delphi (MUTIS), Lisp (Montezuma), Perl (KinoSearch, Plucene), C# (Lucene.Net), C (Lucene4c), C++ (CLucene), Python (PyLucene), Ruby (Ferret) a PHP (Zend Search). [17]

Hlavní přednosti Lucene [17]:

- *Škálovatelnost a vysoce výkonné indexování* - zvládne tak zaindexovat přes 150 GB dat za hodinu s minimálními požadavky na kapacitu RAM (pouze 1 MB halda)
- *Výkon, přesnost a efektivní vyhledávací algoritmus*
  - Využívá ohodnocené vyhledávání, a proto se výsledky řadí podle shody od největší po nejmenší
  - Podporuje řazení podle jakéhokoliv atributu
  - Vyhledávací fráze, dotazy na přibližnou shodu, rozsahové dotazy
  - Podporuje souběžnou aktualizaci indexu a vyhledávání
  - Vícenásobné vyhledávání se spojením výsledků
  - Rychlá a zároveň paměťově efektivní knihovna
- *Meziplatformní řešení* - implementace knihovny v programovacím jazyce Java, která je spustitelná na mnoha platformách

3.3 Solr

Solr je enterprise vyhledávací platforma s otevřeným zdrojovým kódem vycházející z Apache Lucene 3.2. Stejně jako Lucene je i Solr vyvíjen v Javě a první verze byla uvolněna v roce 2004 [19]. Komunikace s touto platformou probíhá prostřednictvím REST API. Dokumenty je možné posílat v různých formátech, jako je JSON, XML, CSV nebo binárně přes protokol HTTP. Odpověď ze serveru je opět v těchto formátech.

Solr je optimalizován pro velkoobjemový provoz, podporuje prostorové vyhledávání, je vysoce škálovatelný, poskytuje administrátorské rozhraní, lze vyhledávat v dokumentech jako je Word nebo PDF, je snadně monitorován a řízen, podporuje rozšíření pomocí pluginů a je flexibilní a jednoduše konfigurovatelný. Nabízí funkce pro pokročilé fulltextové vyhledávání, vytváření návrhů vyhledávacích frází a podporuje analýzu jazyka. [18]

3.4 Elastic Search

Elastic Search je název fulltextového vyhledávače s otevřeným zdrojovým kódem, který vychází z Apache Lucene 3.2. Elastic Search je možné popsat jako distribuované úložiště dokumentů pracující v reálném čase, distribuovaný vyhledávací nástroj s analytikou v reálném čase a také nástroj schopný škálování stovek serverů a práce se strukturovanými i nestrukturovanými daty PB velikostí. Je založen na peer-to-peer [1] architektuře bez SPOF a tudíž při výpadku jednoho serveru nedochází k výpadku celého systému. [1]

Nabízí podporu více jazyků, dokáže vyhledávat na základě geografické polohy, vyhledávat podobné nebo příbuzné záznamy ve stylu „měli jste na mysli“. Toho je docíleno pomocí dotazu MLT [2]. Ke komunikaci s aplikacemi využívá své REST API, kdy téměř každá akce může být provedena prostřednictvím zaslání JSON dokumentu prostřednictvím HTTP. Pro některé programovací jazyky existují knihovny zjednodušující práci.

Elastic Search automaticky vytváří schéma databáze na základě vložených dat, není třeba vytvářet vlastní schéma, je to však možné. Mezi hlavní výhody této technologie patří:

- *Rychlost* - díky tomu je možné na webových stránkách používat filtry, jelikož výsledky vyhledávání jsou k dispozici téměř okamžitě
- *Škálovatelnost* - lze přidávat další servery do clusteru a díky tomu rozložit data optimálně mezi tyto servery
- *Vysoká dostupnost* - Elastic Search detekuje chybové servery a vyřadí je z provozu, data rozdělí mezi zbylé servery a tím zajistí nejvyšší možnou míru dostupnosti
- *Ochrana dat* - při selhání jednoho serveru se práce automaticky převede na další servery jelikož všechny data jsou automaticky replikovány na všechny servery

Portál Gloffer využívá Elastic Search ve verzi 5.6.2, která oproti předchozím verzím především nabízí vylepšení ve formě vysoko-úrovňového REST klienta v Javě. Od února 2018 je již k dispozici verze 6.2.0. Při přechodu na tuto verzi z verze 2.x nebo předchozí je potřeba dokumenty přeindexovat s verzí 5.x, aby bylo možné indexy číst ve verzi 6.x. Dále není možné využívat mnohonásobné mapovací typy pro indexy (lze mít pouze jeden mapovací typ), avšak indexy vytvořené ve verzi 5.x i nadále mohou v nové verzi používat mnohonásobné mapovací typy. V neposlední řadě tato nejnovější verze vyžaduje JDK ve verzi 9. [21]

3.5 Apache HTTP server

Apache HTTP server je multiplatformním softwarovým webovým serverem vyvinutým v 90-tých letech 20. století, který podporuje řadu programovacích jazyků jako je Perl, PHP, Python nebo Tcl. Podporuje také SSL, TLS, proxy modul, přepisování URL adres, konfiguraci souborů

[1] Jednotlivé propojené servery komunikují mezi sebou a není potřeba centrálního prvku.
[2] More Like This dotaz, který pomocí skóre určí, zda se jedná o podobný dokument či nikoliv.

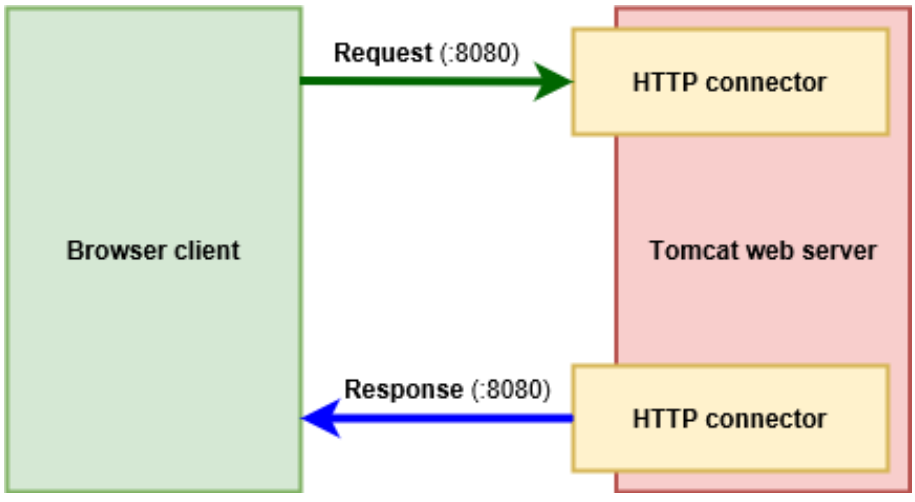
logu a filtraci. Apache dále disponuje externím modulem pro kompresi dat webových stránek posílaných protokolem HTTP. O zabezpečení a prevenci webových stránek před napadením se stará další modul. Disponuje také funkcí virtuálního hostingu, takže na jedné instalaci Apache lze obsluhovat více webových stránek. [42]

Nejnovější verze 2.4 nabízí několik nových modulů a vylepšení [43]:

- Načítání modulů za běhu
- Podpora pro asynchronní zápis a čtení
- Snížení paměťové náročnosti serveru oproti verzi 2.2
- Autentifikace založená na formuláři
- Umožňuje použít stav „session“ pro klienta využitím „cookies“ nebo databázového úložiště
- Umožňuje přidat logování při debuggování v různých fázích zpracování

3.6 Apache Tomcat

Apache Tomcat je webový server a Java servletový kontejner s otevřeným zdrojovým kódem vyvinutý v programovacím jazyce Java. Tato technologie byla vyvinuta roku 1999 a jejím účelem je poskytnout prostředí potřebné k běhu Java EE aplikací zahrnujících Java Servlety a JSP. Umožňuje uživatelům konfigurovat zdroje a využívat XML ke konfiguraci projektů. Způsob zpracování požadavků na portu 8080 je na obrázku 2. Aktuální verze má označení 9.x a oproti předchozí verzi podporuje HTTP/2, OpenSSL a TLS. [23]

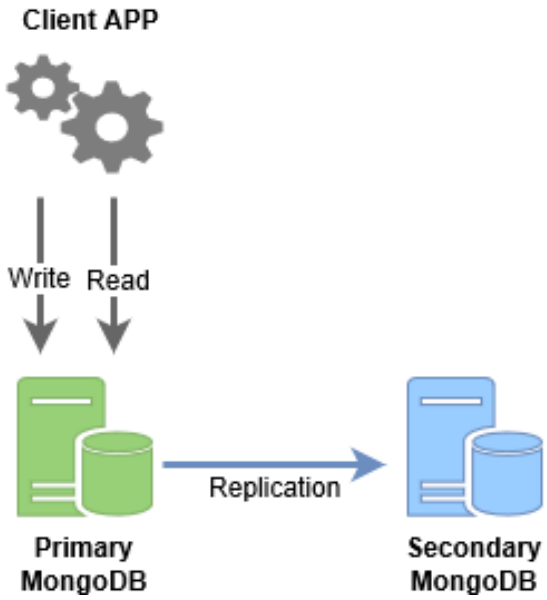


Obrázek 2: Komunikace s Apache Tomcat na portu 8080

3.7 MongoDB

MongoDB je multiplatformní dokumentová NoSQL [3] databáze s otevřeným zdrojovým kódem, která byla poprvé vydána v roce 2009. Důvod vzniku MongoDB databáze je vytvořit datové úložiště, které poskytuje vysoký výkon, vysokou dostupnost a je automaticky škálovatelná. Data se ukládají jako dokumenty ve formátu JSON nebo BSON s dynamickým schématem. Což znamená, že různé dokumenty mohou mít rozdílné vlastnosti a tudíž schéma JSON objektu je rozdílné. Při přidání vlastnosti se pouze přidá daná vlastnost tomuto dokumentu. [6]

Využívá vlastnosti sekundárních indexů, rozsahových dotazů, vyhledávání pomocí regulárních výrazů, řazení, agregací a prostorových indexů. Vysoké dostupnosti dosahuje využitím replikačních sad, kdy se replikační sada skládá ze dvou a více stejných kopií dat. Každá z těchto replikačních kopií je schopná být v roli primární repliky. Pokud primární replika selže, pak replikační sada automaticky nastaví sekundární repliku jako primární. Schéma replikace lze vidět na obrázku 3. Vyvažování vytížení databáze je řešeno pomocí horizontálního škálování, kdy se určí klíč, podle kterého jsou data distribuována mezi shardy [4]. Stejně jako relační databáze umožňuje agregaci dat. Navíc podporuje spouštění javascriptových „poddotazů“ na straně serveru. [6]



Obrázek 3: Replikace MongoDB databáze

Aktuální verze MongoDB s označením 3.6 bude brzy nahrazena verzí 4.0, která umožní transakce dokumentů v replikačních sadách k zajištění konzistence. MongoDB je využíváno firmami jako jsou SAP, Adobe, Ebay, Amazon a dalšími velkými společnostmi. [16]

[3] Databázový koncept, ve kterém se datové objekty ukládají jako oddělené dokumenty uvnitř kolekcí.
[4] Shard se skládá z jednoho master serveru a jednoho až více slave serverů.

3.8 Redis

Redis je NoSQL databáze s otevřeným zdrojovým kódem ukládající dvojice klíč-hodnota, kterou vytvořil Salvatore Sanfilippo v roce 2009. Název je odvozen z „REmote DIctionary Server“ a jedná se o vysoce výkonný server implementovaný v programovacím jazyce C. [12]

Tuto technologii je možné použít jako databázi nebo jako cache pro ukládání výsledků s nastavenou životností záznamů (dále TTL). Pokud Redis používáme jako cache, tak staré data jsou automaticky přepisovány novými daty se stejným klíčem, nebo jsou automaticky smazány po dosažení TTL.

Podporuje řadu datových struktur jako jsou řetězce, hashe, seznamy, množiny, uspořádané množiny s rozsahovými dotazy, bitové mapy, prostorové indexy atd. K zabezpečení vykonání skupiny příkazů v jednom kroku poskytuje transakce.

Mezi hlavní přednosti Redisu patří především rychlé operace vkládání a vyhledání. Vyhledání je rychlé díky faktu, že všechny klíče jsou uloženy v operační paměti. Pokud máme k dispozici dostatečně velkou paměť, jsou v ní uloženy i hodnoty klíčů. Pokud však nemáme dostatek paměti, Redis využívá virtuální paměť a zbytek hodnot zapisuje na disk. Jak jsme již zjistili, Redis je in-memory databáze [5]. Pokud nám dochází volná paměť, pak Redis pro zápis dat na disk se využívá LRU a v novější verzi i LFU algoritmus. V obou případech jde o uložení „nejstarších“ nebo „nejméně používaných“ dat na disk a tím dojde k uvolnění paměti.

Je velmi snadné nakonfigurovat master-slave [6] replikaci, takže slave servery jsou přesnou kopií master serverů a v případě selhání master serveru může být některý ze slave serverů určen za nový master server. Díky master-slave topologii je možné rozložit zátěž mezi více serverů, kdy zapisujeme data na master server a slave servery využijeme ke čtení dat. V průběhu se stará master stará o asynchronní propagaci zápisu dat na slave servery.

Pro komunikaci s touto NoSQL databází existuje spousta Redis klientů v různých programovacích jazycích (C, C#, Haskell, Java, PL/SQL, PHP, Python, R atd.). [25]

3.9 Rabbit MQ

Rabbit MQ je široce využívaný svobodný software s otevřeným zdrojovým kódem pro zprostředkování zpráv, který je implementován v programovacím jazyce Erlang. Jedná se o multiplatformní (Windows, Linux a Unix, Mac OS X) a cloudovou službu (Amazon EC2, Microsoft Azure), která poskytuje klientské knihovny pro mnoho programovacích jazyků (.NET, Java, PHP, Perl, Python, C atd.). Slouží ke koordinaci zpráv aplikací, které mají být integrovány k vytvoření jednoho celku. Tento zprostředkovatel má tedy za úkol zasílat a přijímat zprávy mezi těmito aplikacemi. [13] Způsob koordinace zpráv mezi aplikacemi prostřednictvím RabbitMQ lze vidět na obrázku 4.

[5] Primární úložiště pro data je operační paměť počítače.
[6] Model komunikace kdy jeden server řídí další servery.

Rabbit MQ nabízí tyto vlastnosti [13]:

- *Podpora více protokolů* - komunikace může probíhat prostřednictvím různých protokolů
- *Spolehlivost* - zprávy mohou být trvale ukládány na disk, aby při restartu serveru nedošlo ke ztrátě zpráv
- *Správa prostřednictvím webového uživatelského rozhraní* - lze spravovat uživatele a jejich práva, výměny, fronty
- *Vysoká dostupnost* - je možné vytvořit cluster z několika serverů, kdy se jeví jako jeden zprostředkovatel zpráv, takže při výpadku některého z nich nedojde k selhání zprostředkovatele zpráv
- *Směrování* - předávání zpráv prostřednictvím výměn předtím než jsou uloženy ve frontě
- *Rozhraní příkazové řádky* - stejně jako v případě webového rozhraní jde pomocí příkazové řádky spravovat Rabbit MQ
- *Asynchronní zasílání zpráv* - uživatel nečeká na zpracování požadavku, může pokračovat ve své práci a Rabbit MQ se mezitím postará o zpracování požadavku

Obrázek 4: Schéma využití RabbitMQ

Aktuální verze Rabbit MQ je verze 3.7, která byla vydána v listopadu 2017 a přináší nové vlastnosti jako je Java klient pro RabbitMQ HTTP API. Dále tato verze Rabbit MQ požaduje Erlang ve verzi 19.3 a vyšší. Byly přidány nástroje příkazové řádky pro administraci a jako nová se objevila podpora Proxy protokolu.

3.10 Aerospike

„Aerospike je první pro flash paměť optimalizovaná, vysoce globálně škálovatelná a spolehlivá in-memory NoSQL databáze.“ [15]

Jedná se o databázi s otevřeným zdrojovým kódem vyvinutou v roce 2010, která podporuje flexibilní datové schéma, ACID vlastnosti transakcí a poskytuje vysoký výkon. Tohoto vysokého výkonu je dosaženo díky těmto vlastnostem [15]:

- *Chytrá klientská architektura* - zajišťuje paralelní přístup k více serverům v clusteru a to dovoluje každému klientovi přistupovat k záznamům bez nutnosti přistupovat k centrálnímu prvku, kdy by v případě výpadku tohoto serveru došlo k nefunkčnosti databáze

- *Implementovaná v programovacím jazyce C* - díky tomu nabízí vyšší výkon oproti databázím implementovaným v Javě, jejichž výkon závisí na výkonu JVM a jejím nastavení
- *Hybridní paměťová architektura* - k ukládání využívá jak RAM tak flash paměť

Architektura Aerospike se skládá ze tří vrstev:

- *Klientská vrstva* - klientské knihovny v různých programovacích jazycích (Java, C#, PHP atd.) s Aerospike API
- *Clustery a vrstva distribuce dat* - zajišťuje komunikaci v rámci clusteru, řeší replikaci a synchronizaci dat
- *Vrstva datového úložiště* - vrstva zodpovědná za ukládání dat v RAM a flash paměť

3.11 PHP 7

PHP je skriptovací programovací jazyk s otevřeným zdrojovým kódem vyvinut Rasmusem Lerdorfem v roce 1994, který je určen především pro programování dynamický internetových stránek a aplikací. Lze však tento jazyk použít i k tvorbě desktopových aplikací. Při použití PHP pro webové stránky a aplikace jsou skripty prováděny na straně serveru. Syntaxe tohoto jazyka je ovlivněna několika programovacími jazyky jako je Perl a C. PHP je možné používat na většině operačních systémů (Linux, Unix, Microsoft Windows, Mac OS X atd.) a na webových serverech jako je Apache 3.5 a IIS. PHP podporuje mnoho knihoven např. pro zpracování grafiky, textu, práci se soubory, přístup k databázovým systémům (MySQL, ODBC, Oracle, PostgreSQL, MSSQL). Podporuje také celou řadu internetových protokolů jako je HTTP, SMTP, SNMP, FTP, POP3, IMAP, LDAP atd. [36]

PHP je nejrozšířenějším programovacím jazykem na straně serveru pro vývoj webových aplikací. Přes 80% webových stránek využívá právě jazyku PHP. [44]

PHP 7 je nejaktuálnější verzí PHP, která byla vydána na konci roku 2015. Využívá Zend Engine 3.0, díky němuž poskytuje dvojnásobný výkon s poloviční spotřebou paměti než u verze PHP 5.6. Tento engine je v PHP zodpovědný za [45]:

- Parsování, kontrola syntaxe, kompilaci a provádění PHP skriptů
- Implementaci všech datových struktur v PHP
- Poskytování standardních služeb jako je správa paměti, správa zdrojů atd.
- Komunikaci s moduly pro připojení k externím zdrojům a protokolům (SQL, HTTP atd.)

3.11.1 Rozdíly oproti PHP 5

Hlavní rozdíly mezi novou verzí PHP 7 a předchozí generací PHP 5 jsou následující:

- PHP 7 využívá Zend Engine 3.0
- PHP 7 je přibližně 2x výkonnější než PHP 5 viz [11]
- Přibyly nové vlastnosti (výpis vybraných vlastností) [46]:
  - Deklarace skalárních datových typů parametrů funkcí
  - Deklarace návratových datových typů funkcí
  - Operátor *??* nahrazující funkci *isset()*
  - Operátor *<=>* pro porovnávání dvou výrazů
  - Definování polí pomocí *define()*
  - Unicode syntaxe např. *„\u{9999}“*
  - Použití skupinového *use* pro třídy, funkce a konstanty
  - Funkce pro celočíselné dělení
  - *session_start()* může jako argument přijímat pole pro nastavení sezení (tyto vlastnosti se normálně nastavují v *php.ini* souboru)
- Některé vlastnosti byla označeny za zastaralé [47]:
  - Funkce v PHP 4, která byla používaná jako konstruktor (nyní se musí využívat *__construct()*)
  - Statické volání nestatických funkcí

3.11.2 HHVM

HipHop Virtual Machine je virtuální stroj s otevřeným zdrojovým kódem navržen pro vykonávání programů napsaných v programovacích jazycích Hack a PHP. Používá JIT kompilaci zdrojového kódu k dosažení výkonu, zatímco poskytuje vývojovou flexibilitu, kterou poskytuje PHP. Tvůrcem tohoto virtuálního stroje pro zvýšení výkonu aplikací je firma Facebook, která vývoj na této technologii započala v roce 2010. [22]

HHVM nekompiluje PHP a Hack kód přímo do C++, ale do bajt kódu, který je následně přeložen do strojového kódu dynamicky za běhu JIT kompilátorem. Díky tomu dochází k optimalizaci kódu, které nelze dosáhnout ve statické binární kompilaci. To zajišťuje vysoký výkon aplikací implementovaných v PHP a Hack.

3.11.2.1 Hack

Hack je programovací jazyk, který je založen na jazyce PHP a sdílí s ním velkou část syntaxe a je navržen tak, aby plně spolupracoval s PHP. Hack je použitelný pro vývojovou práci na existujících PHP projektech. Autorem tohoto programovacího jazyku je společnost Facebook. [10]

Hlavní výhodou tohoto jazyku je robustní statická typová kontrola, kdy jsou v průběhu vývoje kontrolovány návratové typy funkcí. Hack dovoluje vytvářet generické třídy a metody, kdy datový typ je přiřazen až v době instanciování třídy nebo volání metody. Podpora lambda výrazů je u tohoto jazyku samozřejmostí, stejně tak podpora asynchronních funkcí. Tato vlastnost nám zajistí, že při čekání na odpověď nám neblokuje zdroje, které mezitím mohou být využity jiným způsobem. Hack nám dovoluje vytvářet vlastní datové typy („typové aliasy“). V neposlední řadě je třeba zmínit XHP, což je vlastnost, která poskytuje přirozenou XML reprezentaci našim HTML výstupům viz 1. Poskytuje nám to ochranu proti XSS útokům a dvojitému „escapování“ znaků. [22]

```
// Bez pouziti XHP
<?hh
$user = 'Jakub';
echo "<tt>Hello $user</tt>";

// S vyuzitim XHP
<?hh
$user = 'Jakub';
echo <tt>Hello {$user}</tt>;
```

Výpis 1: Ukázka využití XHP v praxi

3.11.2.2 Srovnání výkonu PHP 7 a HHVM

Výkonové srovnání PHP 7 a HHVM vychází z prezentace „Speeding up the web with PHP 7“ [11], jejíž autorem je Rasmus Lerdorf. Pan Lerdorf provedl test na serveru s danými parametry viz 1.

Z výkonových testů viz [11] můžeme usoudit, že PHP 7 dosahuje lepších výsledků co se týče zpracovaných požadavků za jednotku času, tak i nižšího zpoždění způsobeného asynchronním vykonáváním kódu. Testovaly se známé webové aplikace jako je Wordpress, Drupal, MediaWiki, OpenCart, phpBB a další.

Téměř ve všech případech zvládlo PHP 7 zpracovat více požadavků než stejná aplikace s využitím HHVM. Stejně tak i zpoždění je lepší v případě PHP 7. Někdy rozdíly znamenaly i o téměř 30% více zpracovaných požadavků za jednotku času ve prospěch PHP 7.

Tabulka 1: Parametry testovacího serveru pro srovnání výkonu HHVM a PHP 7 [11]

Procesor	Gigabyte Z87X-UD3H i7-4771 4 cores @ 3.50GHz
Operační paměť RAM	16GB 1600MHz
Pevný disk	Toshiba THNSNHH256GBST SSD
Operační systém	Linux debian 3.16.0-4-amd64
Databáze	MySQL 5.6.24
Webový server	nginx-1.6.2
Síť	100Mbps

V době, kdy se používalo PHP 5.x se HHVM vyplatilo implementovat, jelikož zaručuje několikanásobně vyšší počet zpracovaných požadavků za jednotku času a menší zpoždění u všech testovaných aplikací. Avšak v době používání PHP 7 se využívání technologie HHVM vyplatí jen pro některé případy použití (Wordpress a MediaWiki). Je potom spíše otázkou testování, která technologie se pro naší aplikaci hodí více.

3.12 Nette Framework

„Nette je sada samostatných PHP 7 komponent tvořících dohromady framework, vyhodnocený jako 3. nejpopulárnější na světě. Nette klade mimořádný důraz na produktivitu, čistý kód a bezpečnost.“ [27]

Nette Framework je moderní framework s otevřeným zdrojovým kódem vyvinutý Davidem Grudlem. Slouží k vytváření webových aplikací jako jsou e-shopy, blogy, redakční systémy a jiné. Obsahuje komponenty, které slouží k ladění, testování, vytváření bezpečných webových aplikací. Pro zajištění bezpečnosti poskytuje nástroje k ochraně proti útokům jako jsou XSS, CSRF, podstrčení a ukradení session. Další podstatnou výhodou Nette je rozsáhlá komunita uživatelů v ČR, která velmi obětavě dokáže pomoci na fóru s řešením problémů.

Pomocí šablonovacího systému Latte, který je součástí frameworku, jsme schopni rychle a bezpečně vytvořit šablonu. Ta se automaticky přeloží na HTML výstup, který je navíc zabezpečen před XSS útoky. Další vlastností je využití formátu NEON, který slouží pro serializaci dat. Tento formát je podobný formátu YAML s tím rozdílem, že syntaxe NEON formátu je jednodušší a rychlejší při parsování. Tracy neboli „Laděnka“ slouží programátorům k odhalení chyby a jejímu logování. Dále pak díky ní můžeme vypisovat proměnné a měřit výkonové nároky na server. Součástí jsou také třídy pro práci s formuláři, databázemi, pomocné třídy pro práci s obrázky, řetězci, souborovým systémem, ale i pro parsování JSON souboru, třídy pro automatické načítání tříd nebo pro generování tříd. Tester slouží k automatickému testování webové aplikace.

Nette nabízí rozšíření pro vývoj aplikací, ať už jde o propojení se sociálními sítěmi, bankovními bránami, komunikaci s databázovými systémy atd. Pro implementaci webových aplikací si jej vybraly společnosti jako jsou CSFD, GE Money, DHL, ESET atd. [27]

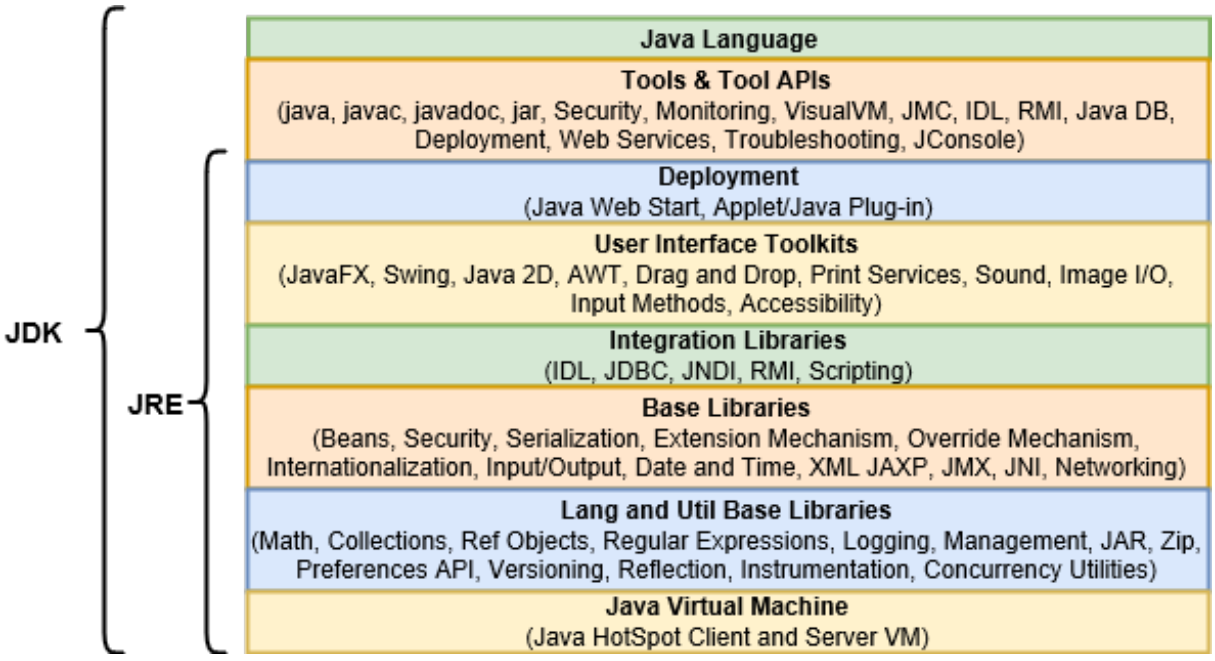
3.13 Java

Java je vysoko-úrovňový objektově orientovaný programovací jazyk vytvořen firmou Sun Microsystems v roce 1995. Velkou výhodou Javy oproti konkurenci je fakt, že program napsaný v Javě je spustitelný na jakékoliv platformě. Program je kompilován do nezávislého bajt kódu a ten je interpretován díky virtuálnímu stroji (JVM) do přirozených strojových instrukcí dané platformy, na které je program spouštěn. [9]

Další předností tohoto programovacího jazyka je práce s vlákny, kdy jsme schopni dosáhnout využitím více vláken rychlejšího zpracování a tím vyšší propustnosti systému. Garbage collector se stará o uvolňování přidělované paměti a programátor, tak nemusí řešit vytváření destruktorů objektů jako v případě programovacího jazyka C/C++. Dále poskytuje nástroje pro zachytávání a vyvolávání výjimek a silnou typovou kontrolu. Bezpečnost Javy je zajištěna díky JVM, která tvoří speciální vrstvu mezi programem a OS.

Rozlišujeme tyto vývojářské sady (více v obrázku 5):

- **JIT** - kompilátor překládá bajt kód do přirozených strojových instrukcí za běhu programu
- **JVM** - abstraktní výpočetní stroj umožňující běh Java aplikací
- **JRE** - softwarový balík vyžadovaný k běhu Java aplikací, který zahrnuje JVM
- **JDK** - JRE a nástroje pro vývoj Java aplikací jako jsou *javac* kompilátor, *javadoc* pro automatické vytváření dokumentace programu, *java* pro zavádění Java aplikací atd.



Obrázek 5: Členění JDK

Platformy [38]:

- **Java SE** - poskytuje základní funkcionalitu programovacího jazyka Java, jako jsou základní datové typy, objekty, dále pak třídy pro práci se sítěmi, bezpečností, databázovým přístupem, vývojem grafického uživatelského rozhraní a parsování XML
- **Java EE** - poskytuje funkcionalitu platformy Java SE a navíc API a prostředí pro vývoj škálovatelných, spolehlivých a bezpečných síťových aplikací
- **Java ME** - poskytuje API a zjednodušený JVM pro běh aplikací na malých zařízeních jako jsou mobily, toto API je podmnožinou Java SE se speciálními knihovnami použitelnými pro vývoj mobilních aplikací, většinou jsou Java ME aplikace klienty služeb Java EE platformy
- **JavaFX** - platforma po vytváření internetových aplikací, které mohou sloužit jako klienti služeb Java EE platformy

Posledním vydáním JDK je verze 10, která byla vydána v březnu 2018. Ta k předchozí verzi přidává možnost odvození datového typu lokální proměnné viz 2. Další novou vlastností je rozhraní Garbage collectoru umožňující vývojářům řídit uvolňování paměti, dále pak plně paralelní Garbage collector pro zrychlení běhu aplikací. Přináší také možnost zastavit jednotlivé vlákna. [40]

```
// JDK 10
var list = new ArrayList<String>(); // odvodi datovy typ ArrayList<String>

// JDK x; x < 10
ArrayList<String> list = new ArrayList<String>();
```

Výpis 2: Ukázka odvození datového typu lokální proměnné

3.13.1 Java EE 8

Poslední verze Java EE byla vydána ve druhé polovině roku 2017 a poskytuje nové vlastnosti, jako je podpora HTTP/2 [7] v servletových aplikacích, REST reaktivní klientské API, nové bezpečnostní API, podporu událostí zasílaných serverem a další. [39]

3.14 Spring Framework

Spring Framework je aplikační framework vyvinutý roku 2003, jehož autorem je pan Rod Johnson. Tento pán vydal framework v první verzi necelý rok po vydání vlastní knihy *Expert One-on-One J2EE Design and Development*, ve které popisuje problémy týkající se vytváření J2EE aplikací. Důvodem vzniku frameworku je usnadnění vývoje enterprise aplikací. [28]

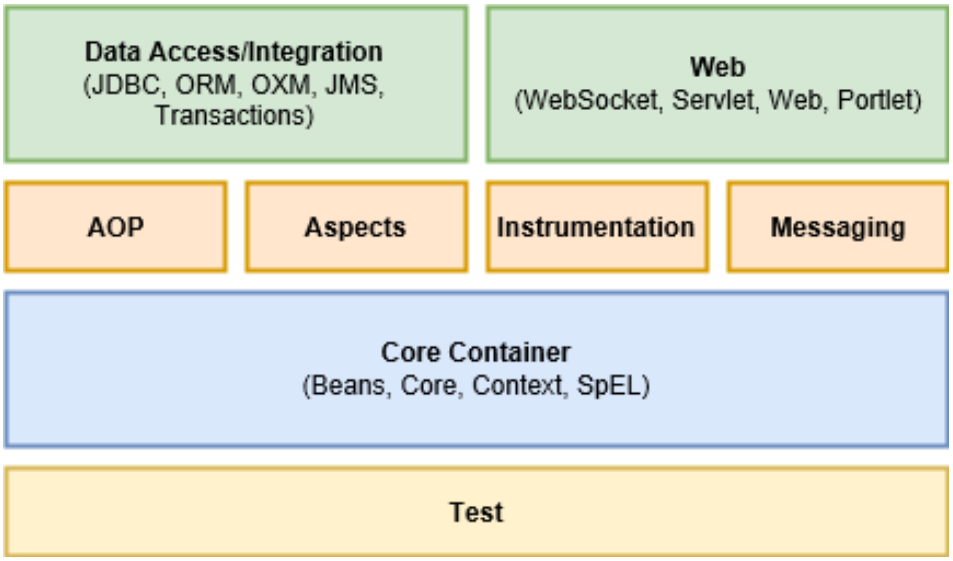
[7] Nová verze HTTP, která přináší optimalizaci přenosu použitím datových proudů. Navíc podporuje multiplexing a posílání dat ihned při prvním dotazu.

Tento framework slouží pro vytváření Java aplikací, ať už jde o klasické, webové nebo J2EE aplikace. Jeho výhodou oproti klasickému vývoji J2EE aplikací spočívá v tom, že umožňuje vytváření aplikací z POJO (Plain Old Java Objects). [30] Díky tomu nejsme nuceni vytvářené třídy rozšiřovat nebo implementovat jiné třídy nebo rozhraní Spring API. Tím se výsledná implementace zjednoduší a zpřehlední. Ukázka využití POJO viz 3.

```
// without POJO
public class ExampleListener implements MessageListener {

    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            try {
                System.out.println(((TextMessage) message).getText());
            } catch (JMSException ex) {
                throw new RuntimeException(ex);
            }
        } else {
            throw new IllegalArgumentException("Message must be of type
                TextMessage");
        }
    }
}

// with POJO
@Component
public class ExampleListener {

    @JmsListener(destination = "myDestination")
    public void processOrder(String message) {
      System.out.println(message);
    }
}
```

Výpis 3: Ukázka zjednodušení implementace využitím POJO [30]

Framework ve svém jádře využívá návrhového vzoru Inversion of Control, který má za úkol přesunout zodpovědnost za vytváření a provázaní objektů z aplikace na framework. Díky tomuto návrhovému vzoru se zvýší modularita a rozšiřitelnost vyvíjené aplikace. Mezi další vlastnosti frameworku patří podpora konfigurací pomocí XML a anotací, MVC pro vývoj webových aplikací, jednoduché aplikační testování atd. [28]



Obrázek 6: Architektura Spring Frameworku

Spring Framework je organizovaný do modulů rozdělených do skupin viz 6. Při vývoji lze tyto moduly využít samostatně. Moduly jsou navzájem odděleny a tudíž není třeba využívat úplně všech, když je nepotřebujeme.

Aktuální verze frameworku má označení 5.0.x a je k dispozici od druhé poloviny roku 2016. Hlavní rozdíl oproti předchozí verzi je nutnost použití JDK 8+ a Java EE 7+. Nová verze podporuje funkcionální programování prostřednictvím programovacího jazyku zvaného Kotlin[8] a přichází s vylepšeními pro testování vyvíjené aplikace. [29]

3.15 Apache Hadoop

Hadoop je multiplatformní (Windows, Linux, Mac OS X atd.) framework s otevřeným zdrojovým kódem, který byl navržen pro spolehlivé a distribuované zpracování velkých dat v rámci clusterů sestavených z počítačů. Škálovatelnost tohoto frameworku je vysoká, tzn. je možné využít jeden až tisíce serverů. Hadoop se skládá z těchto modulů: [32]

- **Hadoop Common** - společné nástroje sloužící k podpoře dalších Hadoop modulů
- **Hadoop Distributed File System** - distribuovaný souborový systém poskytující přístup k datům aplikací
- **Hadoop YARN** - framework pro plánování úloh a řízení zdrojů clusteru
- **Hadoop MapReduce** - systém pro paralelní zpracování velkých dat

Dalšími Hadoop projekty jsou například HBase 3.16 a Cassandra 3.17.

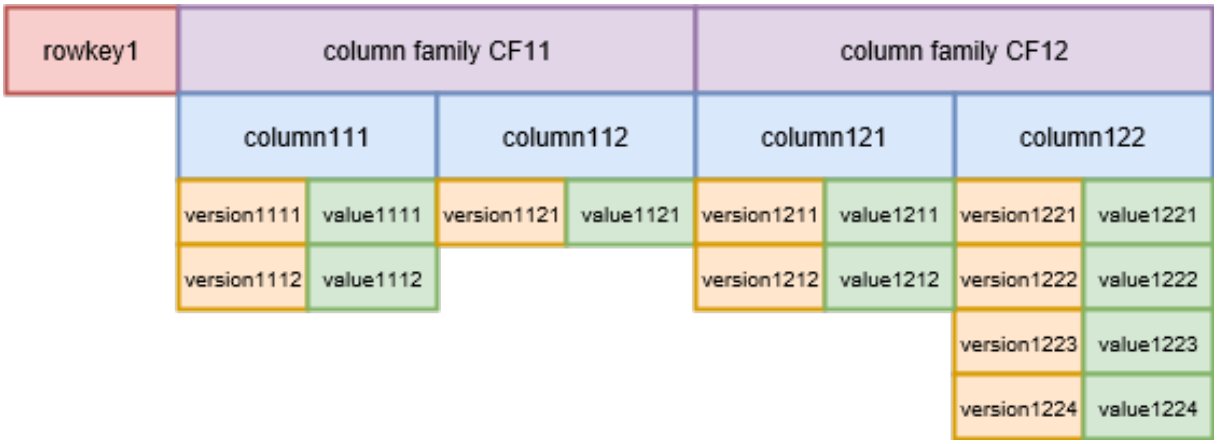
[8] Kotlin je staticky typovaný programovací jazyk pro vývoj multiplatformních aplikací.

Hadoop je používán spoustou velkých firem jako je Yahoo!, kde jejich největší clustery jsou tvořeny 4 500 servery a jsou využívány k webovému vyhledávání. Dále pak Facebook, který tuto technologii používá k ukládání logů, prostorových dat a následnému strojovému učení a analýze. Využívají clustery o velikosti až 1 100 serverů s úložištěm okolo 12 PB. [33]

3.16 Apache HBase

HBase je NoSQL distribuovaná sloupcově orientovaná databáze s otevřeným zdrojovým kódem určená pro ukládání velkých dat, která vznikla v roce 2010 jako součást projektu Hadoop 3.15, avšak vývoj započal roku 2006. Implementace této technologie probíhala v programovacím jazyce Java. Tato databázová technologie je navržena tak, aby ukládala velké tabulky (miliardy řádků s miliony sloupců) a přitom garantovala nízké zpoždění a náhodné čtení a zápis téměř okamžitě. HBase technologie byla navržena tak, aby poskytovala vysokou dostupnost a dokázala se rychle automaticky obnovit po pádu serveru. [7]

Datový model HBase databáze se skládá z tabulek obsahujících řádky, kde jsou data organizována do rodin sloupců (seskupuje sloupce v každém řádku). Při přístupu ke klíči tedy máme přístup k seznamu rodin sloupců. Tyto rodiny sloupců obsahují odkazy na seznamy sloupců. Jednotlivé sloupce poté obsahují seznamy časových razítek, která se dále mapují na samotné hodnoty. Způsob uložení záznamu lze vidět na obrázku 7.



Obrázek 7: Způsob uložení záznamu v HBase

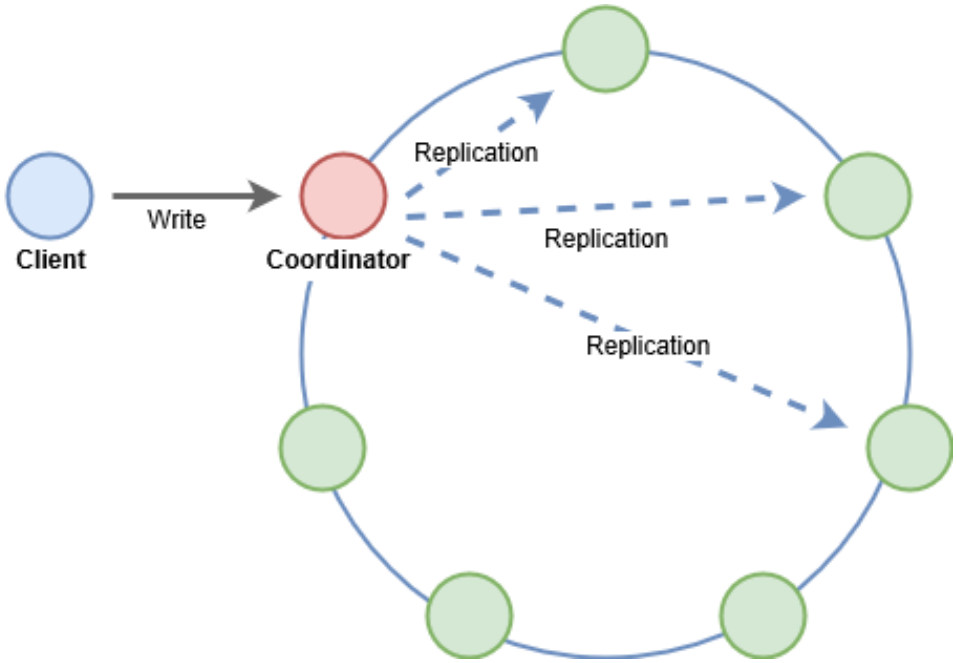
Tato technologie poskytuje mechanismy pro replikaci dat, umožňuje automatické vyvažování výkonu mezi shardy, filtrování a zpracování na straně serveru. Pro práci s HBase je k dispozici Java API, nebo HBase externí REST API, C/C++ Apache HBase klient, dále pak Scala a Jython klienti. [35]

Firma Facebook používá HBase databázovou technologii pro ukládání zpráv. Museli učinit rozhodnutí, zda využít MySQL 3.1, HBase nebo Cassandru 3.17. MySQL neměla dostatečný výkon při růstu indexů a Cassandra měla složitý model. [34]

3.17 Apache Cassandra

Cassandra je NoSQL distribuovaná databáze s otevřeným zdrojovým kódem, která nabízí škálovatelnou architekturu. Ta nabízí možnost přidávání více serverů do databázového clusteru. Data jsou poté replikována na tyto servery. Jednotlivé servery jsou v podstatě master i slave zároveň, tudíž se jedná o architekturu bez SPOF. Avšak každý uzel ví o všech ostatních, tudíž klient přistupuje ke clusteru jako k jedné entitě a nemusí si pamatovat, kde jsou data uložena. Navíc poskytuje robustní rozhraní pro modelování a dotazování dat. K tomuto účelu využívá primárně CQL, což je jazyk podobný klasickému SQL. [8]

Tato technologie nabízí díky automatické replikaci na více uzlů dostupnost a trvalost uložení dat. Uzly, které selžou můžou být nahrazeny za běhu a nedochází tak k nedostupnosti služeb. Diagram znázorňující replikaci dat lze vidět na obrázku 8. Na rozdíl od tradičních relačních a dokumentových databází, které jsou optimalizovány na výkon čtení, je Cassandra optimalizována na výkon zápisu. S daty pracuje takovým způsobem, že nemodifikuje data na disku, ale vytvoří nový záznam při aktualizaci.



Obrázek 8: Prstencová architektura Apache Cassandra zajišťující replikaci dat

Další vlastností Cassandry je používání sekundárních indexů. Díky tomu lze jednoduše vyhledávat nad atributy, které nejsou primárním klíčem. Dále poskytuje efektivní uspořádání výsledku, kde tabulky mohou být uspořádány podle určitých sloupců.

Cassandra poskytuje ovladače pro programovací jazyky jako je Java, Python, .NET, PHP, C++, Haskell, Perl atd. Jejich služeb využívají společnosti jako jsou Apple, CERN, eBay, GitHub, Reddit, Instagram atd. [37]

3.18 Google Bigtable

Google Bigtable je vysoce výkonná NoSQL databáze, která se objevila v roce 2005. Jedná se o cloudovou [9] službu zajišťující trvale nízkou dobu odezvy s vysokou propustností. Tato technologie je škálovatelná až na stovky PB zcela automaticky. Navíc všechny data šifruje a tím zajišťuje jejich bezpečnost. S bezpečností navíc souvisí také řízení přístupu k uloženým datům. [26]

Tato služba se vyznačuje tím, že je nabízena jako plně spravovaná služba a tím je koncový uživatel odstíněn od potřeb konfigurace a ladění výkonu. Data jsou ukládána redundantně pro potřeby zálohy, není třeba konfigurovat další úložiště a platit další poplatky za ukládání redundantních dat.

Přidávání uzlů do clusteru Google Bigtable se děje bez potřeb restartování služby a je zajištěno automatické rozložení dat na tyto uzly. Součástí je API zajišťující přenositelnost dat mezi HBase 3.16 a Bigtable. Samozřejmostí je dostupnost dat odkudkoliv, kde má uživatel přístup k internetu.

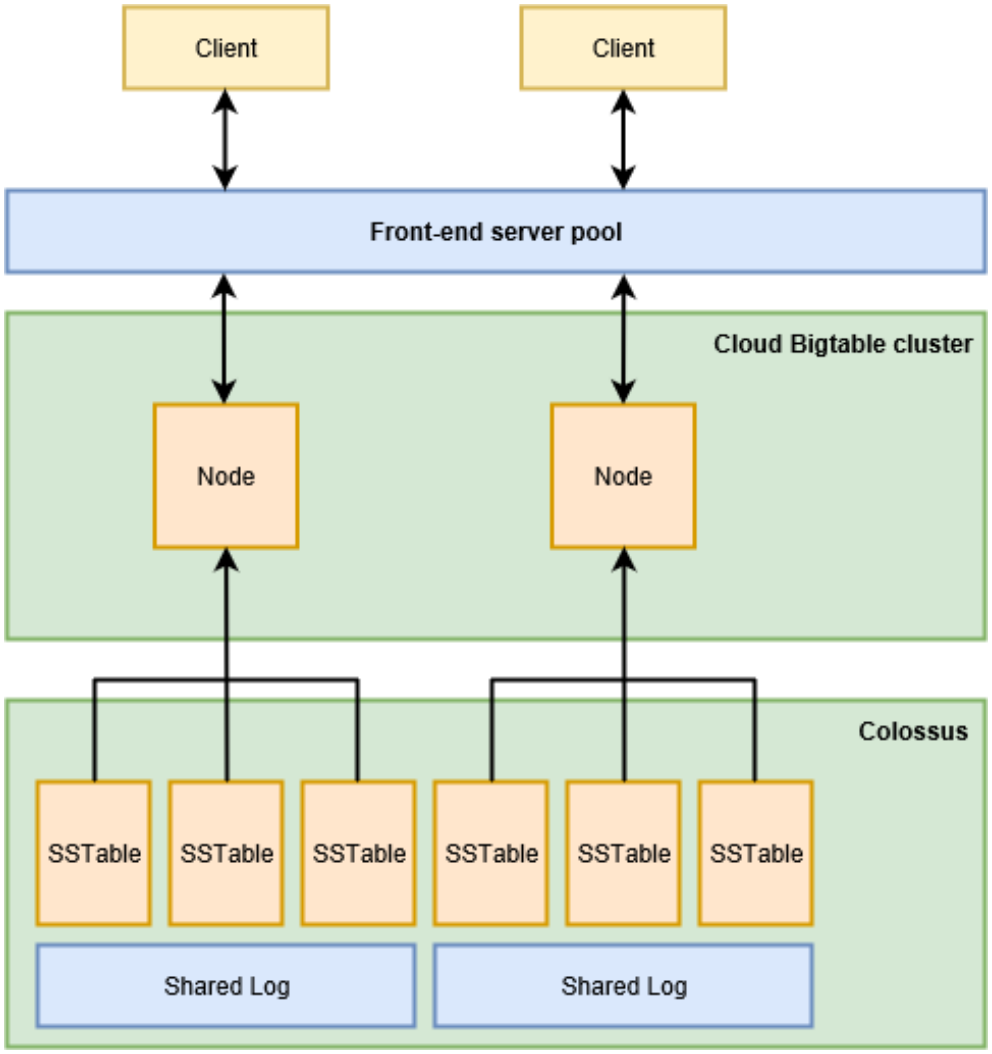
Data jsou uložena ve škálovatelných tabulkách, kde jednotlivé řádky popisují jednotlivé entity a sloupce obsahují hodnoty pro každý řádek. Na tento typ reprezentace dat jsme zvyklí z klasických relačních databází. Avšak související sloupce jsou seskupovány dohromady a tvoří tzv. „sloupcovou rodinu“. Každý sloupec je pak identifikován kombinací „sloupcové rodiny“ a sloupcového kvalifikátoru, což je unikátní jméno v rámci této rodiny.

Průniky řádků a sloupců mohou obsahovat více buněk s různým časovým razítkem, což nám udává změny záznamu v čase, jak byly měněny. Pokud buňky neobsahují žádné data, nezabírají žádný prostor.

Architekturu této technologie popisuje obrázek 9. Klienti posílají požadavky prostřednictvím „front-end“ serveru, který dále požadavky přeposílá jednotlivým uzlům. Tyto uzly jsou organizovány do jednotlivých clusterů, kde cluster představuje instanci Bigtable. Jednotlivé tabulky jsou rozděleny do bloků nazývaných „tablety“ k vyvážení pracovní zátěže dotazů. Tyto „tablety“ ve formátech SSTable jsou uloženy v souborovém systému společnosti Google zvaném „Colossus“. SSTable je perzistentní uspořádaná mapa klíčů a hodnot, kde obojí jsou řetězce bytů. Jednotlivé tablety souvisí s konkrétním uzlem Bigtable. Všechny zápisy do SSTable souborů jsou uloženy ve sdíleném logu. [31]

Bigtable služba je placená hodinovou sazbou za uzly, dále pak měsíčně za GB úložiště (HDD, SSD). Poslední účtovanou částkou je přenos dat z Bigtable k uživateli po síti závisející na lokaci uživatele. Detailnější ceník je možné vidět v tabulce 2.

[9] Model založený na internetu, kdy jsou služby, data a programy serverů přístupné uživatelům vzdáleně např. pomocí internetového prohlížeče.



Obrázek 9: Architektura Google Bigtable

Tabulka 2: Ceník služeb Google Bigtable [26]

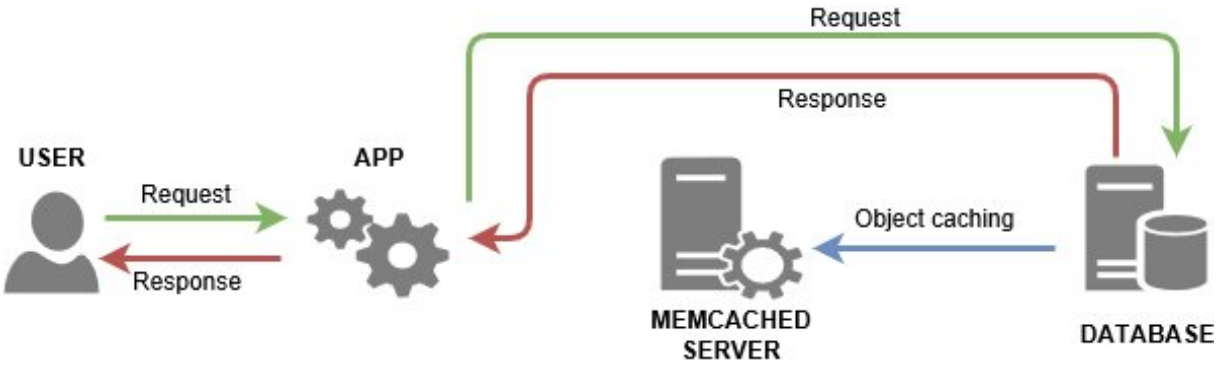
Vlastnost	**Sazba v $**	**Účtováno**
Uzel	0,650	hodina
SSD úložiště (GB)	0,170	měsíc
HDD úložiště (GB)	0,026	měsíc
Přenos po sítí (příchozí)	0,000	GB
Přenos po sítí (odchozí)	0,080 - 0,230	GB

3.19 Memcached

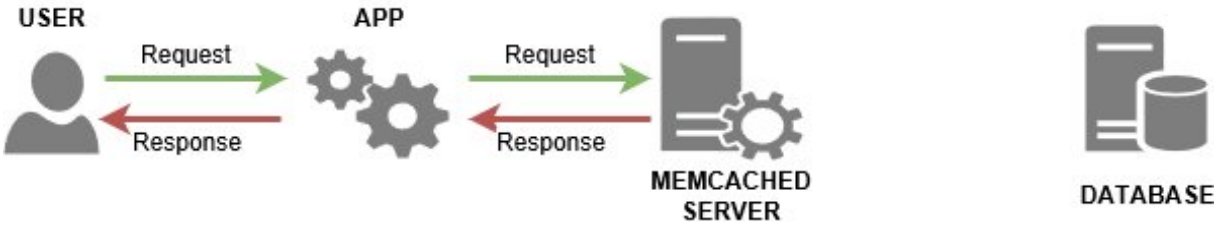
Memcached je vysoce výkonný distribuovaný paměťový cachovací systém s otevřeným zdrojovým kódem využívaný ke zrychlení dynamických webových aplikací díky snížení potřeb načítání databáze. Je to in-memory [10] úložiště uchovávající dvojice klíč-hodnota nebo malé data jako jsou řetězce a objekty uchovávající výsledky „vyrendrovaných“ stránek nebo databázových a API volání. [24]

Memcached funguje na principu přidělování nevyužitých částí paměti systému těm aplikacím, které je potřebují. Bez této technologie mají webové aplikace přidělenou paměť a nelze využívat celkové kapacity paměti serveru.

Na obrázku 10 lze vidět zpracování dotazu během prvního dotazu, kdy data nejsou uložena v Memcached. Při tomto provádění dotazu se data z databáze uloží do operační paměti Memcached serveru. Při dalším volání stejného dotazu se však data nezískávají z databáze, ale z Memcached serveru viz 11.



Obrázek 10: Cachování dat pomocí Memcached



Obrázek 11: Použití dat uložených v Memcached

Záznamy jsou na serveru uloženy jako klíč, expirační čas, volitelně nějaký příznak a samotná data. Polovina implementační logiky systému je vykonávána klientem (např. vybírá, který server čte nebo zapisuje záznamy), druhá polovina serverem (způsob uložení a přístupu k záznamům atd.). Mezi další vlastnost Memcached se řadí fakt, že servery nejsou navzájem propojeny, tudíž neprobíhá žádná synchronizace, zasílání zpráv ani replikace dat. Přidáním serverů se však zvyšuje dostupná paměť.

[10] Zpracování dat přímo v operační paměti počítače.

Využívá LRU algoritmu pro odložení starých dat na disk při nedostatku operační paměti. Všechny příkazy jsou implementovány tak, aby byly provedeny co nejrychleji. Mají konstantní asymptotickou složitost.

Aktuální verze vydána v prosinci 2017 podporuje externí flash paměť pro rozšíření cachovacího úložiště a je zdarma ke stažení na webu `https://memcached.org/`. API této cachovací technologie je dostupné pro většinu programovacích jazyků.

3.20 Webové služby

„Webová služba je rozhraní aplikační funkcionality přístupné přes síť, které je vybudováno využitím standardních internetových technologií.“ [14]

3.20.1 REST

REST je architektonický styl, který má tyto vlastnosti:

- Využívá HTTP
- Musí se jednat o klient-server systém
- Každý požadavek by měl být nezávislý na dalších požadavcích
- Je možné využít cachovací technologie
- Každý zdroj musí mít unikátní adresu k přístupu (URI)
- Využívá metody pro přístup ke zdrojům
  - **POST**- slouží k vytvoření záznamu, jako návratová hodnota se většinou posílá vytvořené ID záznamu
  - **GET** - slouží k navrácení kolekce nebo jednoho záznamu
  - **PUT** - slouží k požadavku na výměnu kolekce nebo jednoho záznamu za jinou kolekci nebo záznam (pokud kolekce nebo záznam neexistuje, pak dojde k vytvoření)
  - **DELETE** - slouží ke smazání kolekce nebo jednoho záznamu

Díky těmto a dalším vlastnostem je aplikace s využitím REST architektury udržovatelná a rozšiřitelná.

3.20.2 SOAP

SOAP je aplikační komunikační protokol, který slouží k výměně zpráv založených na XML. Má tyto vlastnosti:

- Podporuje pouze XML datové formáty

- Kromě HTTP může využívat i jiné transportní protokoly
- Zajišťuje ACID vlastnosti transakcím
- Poskytuje možnost asynchronního zpracování

3.20.3 Srovnání REST a SOAP

Tabulka 3: Srovnání REST a SOAP

REST	**SOAP**
Architektonický styl	Protokol
Může využívat SOAP služby	Nemůže použít REST služby
Může využívat cachování	Nelze využít cachování
Podpora různých datových formátů (HTML, XML, JSON atd.)	Pouze XML
Používá URI	Používá rozhraní služeb
Jednodušší na použití	Složitější na použití
Podpora SSL	SSL + další zabezpečení
Transakce „bez ACID“	ACID transakcí
Nejedná se o standard	Standardizován

4 Návrh

Při návrhu nástroje pro sjednocení datových zdrojů bylo zapotřebí učinit rozhodnutí týkající se architektury rozhraní, logiky zpracování dat, funkcionality systému, použitých a dostupných technologií. Hlavní myšlenkou navrhovaného systému je jednotný přístup k datům ať už prostřednictvím mobilních aplikací, webového prohlížeče, případně přes desktopovou aplikaci (administrace systému).

Bylo potřeba učinit rozhodnutí, jakého programovacího jazyku využít při implementaci tohoto datového rozhraní. Aplikace má komunikovat prostřednictvím REST API a má poskytovat vysoký výkon. Rozhodoval jsem se, zda použít PHP 3.11 nebo Java 3.13. Z důvodu poskytnutí vyššího výkonu, možnosti škálování a jednoduchosti implementace REST API byla zvolena Java s frameworkem Spring. Datové zdroje zůstaly stejné jako v případě stávajícího webového portálu (MySQL 3.1, MongoDB 3.7, Elastic Search 3.4, Redis 3.8), aby byl zajištěn souběh obou aplikací během prvotního spuštění nástroje pro sjednocení datových zdrojů.

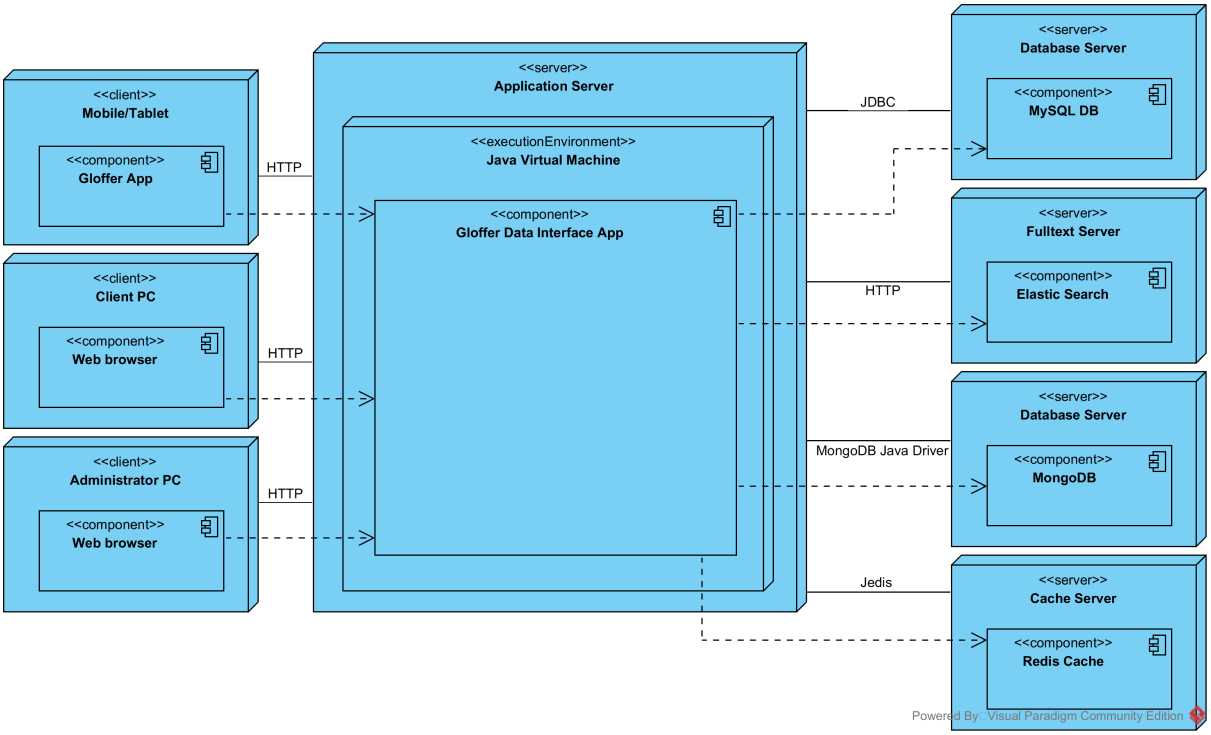
4.1 Architektura rozhraní

Architektura navrhovaného rozhraní je příkladem klient-server aplikace, kdy uživatel zašle požadavek na server. Server zpracuje požadavek a vrátí data uživateli jednotným způsobem, ať už se jedná o mobilní, webovou nebo desktopovou aplikaci.

Fyzickou architekturu vyvíjeného rozhraní nejlépe popisuje diagram nasazení viz 12. Komunikace mezi datovým rozhraním a mobilní aplikací nebo počítači klientů popř. administrátorským počítačem probíhá prostřednictvím HTTP protokolu. Využívá se zde REST API datového rozhraní. Toto rozhraní poté komunikuje se čtyřmi oddělenými servery, které slouží jako datové zdroje. Komunikace mezi datovým rozhraním a datovými zdroji probíhá pomocí protokolu JDBC v případě databáze MySQL, dále pak využívá HTTP protokol pro komunikaci s Elastic Search databází. Pro komunikaci s MongoDB je využito MongoDB ovladače určeného pro programovací jazyk Java. Komunikace s Redisem probíhá prostřednictvím Jedis konektoru.

Obsah jednotlivých databázových systémů:

- **MongoDB** - „feedové“ produkty
- **Elastic Search** - obsahová část (katalogové produkty a „feedové“ produkty)
- **MySQL** - obsahová část (uživatelé, firmy, obchody, poptávky, nabídky atd.)
- **Redis** - cachované výsledky dotazů a často sestavované výsledky

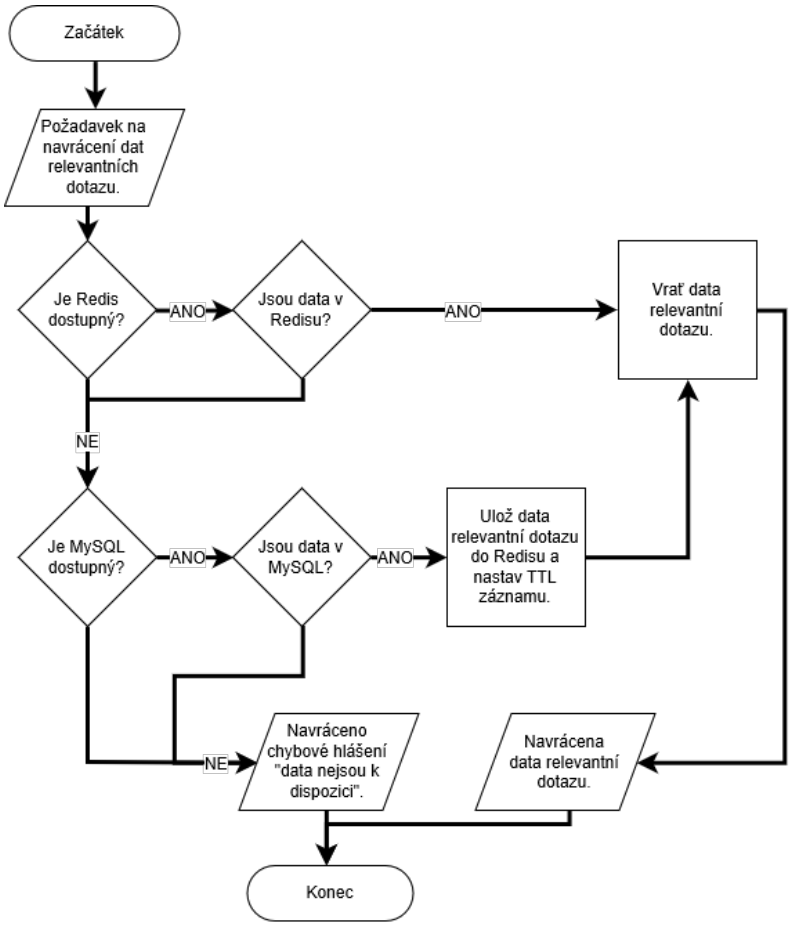


Obrázek 12: Diagram nasazení rozhraní pro sjednocení datových zdrojů

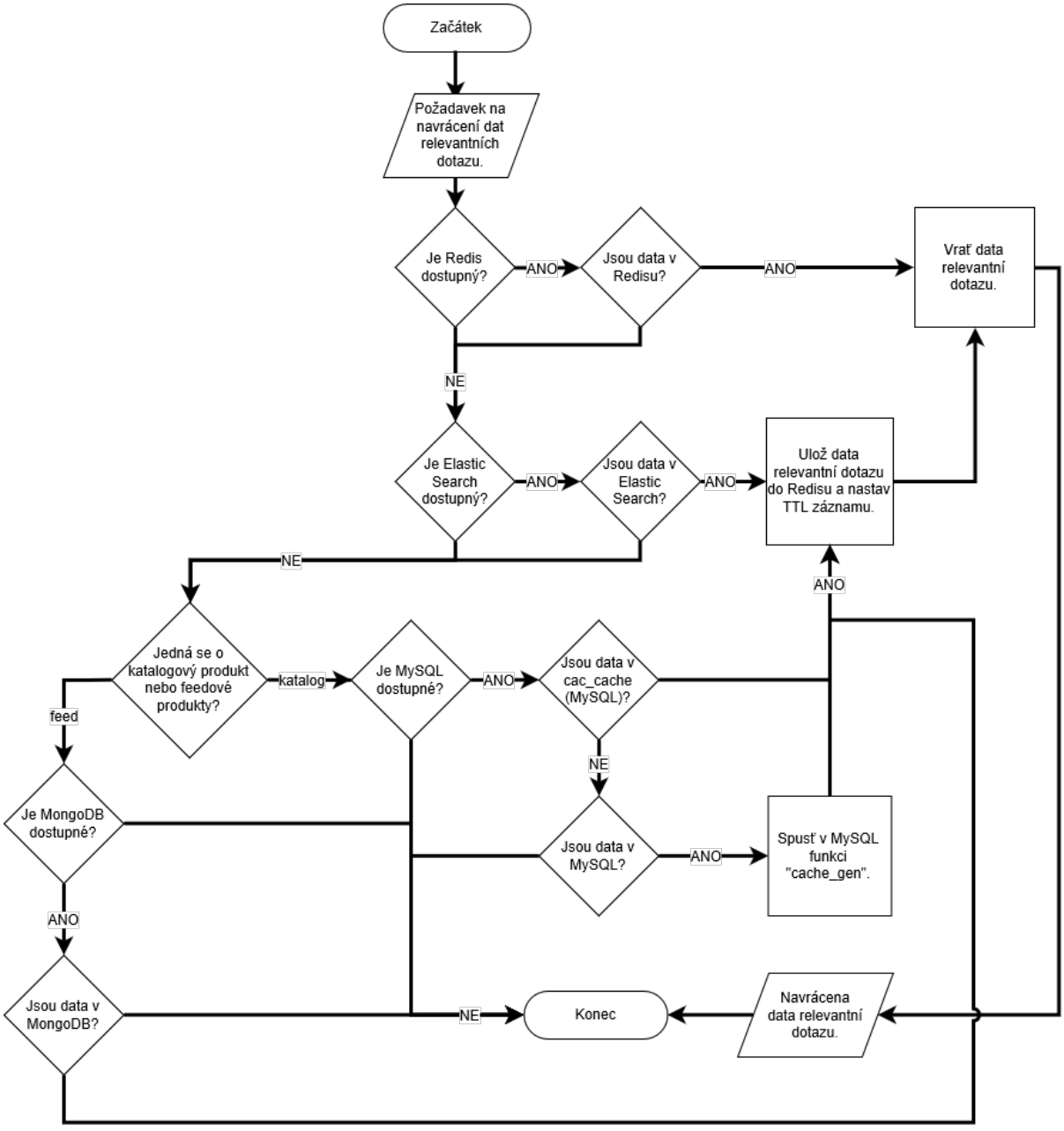
4.2 Logika zpracování dat

Samotná logika zpracování dotazů závisí na tom, o jaké data se jedná. Pokud potřebujeme pracovat s daty uživatele, firmy nebo e-shopu, využívá se pouze databáze MySQL a poté jsou výsledky z tohoto datového zdroje uloženy v cache Redis. Vývojový diagram popisující průběh zpracování dotazu je na obrázku 13.

Zpracování dotazu pro získání informací o produktech je však komplikovanější. V prvotním kroku tyto data hledáme v cache Redis, pokud však data nejsou „nacachována“, je potřeba hledat katalogový produkt v Elastic Search databázi a nabídky jednotlivých e-shopů souvisejících s těmito produkty v Elastic Search případně MongoDB (pokud by Elastic Search selhal nebo data neobsahoval). Pokud tyto data nejsou k dispozici ať už díky nedostupnosti Elastic Search popř. MongoDB, nebo data nebyla do těchto databází importována, tak projdeme „cac_cache“ v MySQL. V poslední fázi, kdy data nejsou ani v „cac_cache“, je potřeba použít funkci „cache_gen“ v MySQL databázi, která nám požadovaný objekt sestaví a uloží do „cac_cache“. Výsledek je opět uložen do cache Redis. Tento průběh zpracování zachycuje vývojový diagram viz 14.



Obrázek 13: Vývojový diagram znázorňující průběh získávání dat kategorie, uživatele a firmy z datových zdrojů



Obrázek 14: Vývojový diagram znázorňující průběh získávání dat produktu z datových zdrojů

4.3 Funkcionalita systému

Systém by měl být schopný poskytovat tyto operace:

- Vyhledávání produktů (katalogových i „feedových“) a jejich uspořádání podle ceny nebo názvu
  - Parametrické vyhledávání
  - Fulltextové vyhledávání
  - Vyhledávání v kategoriích
- Vykonávání specifického dotazu nad zvoleným datovým zdrojem
- Aktualizace cache Redis na požadavek viz 5.11
- Automatické prodlužování životnosti záznamu Redisu viz 5.10
- Získávání dat z datového zdroje, který je aktuálně dostupný viz 5.9
- Deduplikace záznamů v Redisu viz 5.12
- Při nedostupnosti některého datového zdroje tento zdroj nějakou dobu nevyužívat viz 5.9
- Registrace uživatele
- Výpis uživatele/firmy/e-shopu
- Práce související s poptávkami, nabídkami a s tím svázanými diskusemi
- Generování výsledného JSON dokumentu popisujícího uzavření obchodní smlouvy

Funkcionalitu popisuje diagram případu užití viz 15.

V rámci funkcionality aplikace by měly být zajištěny tyto operace (popsáno pomocí DML jazyka SQL) v rámci jednotlivých datových zdrojů:

- **Elastic Search**: SELECT

  Pouze vracíme data z tohoto datového zdroje. O vkládání, aktualizaci a mazání dokumentů se stará jiná aplikace.

- **MongoDB**: SELECT

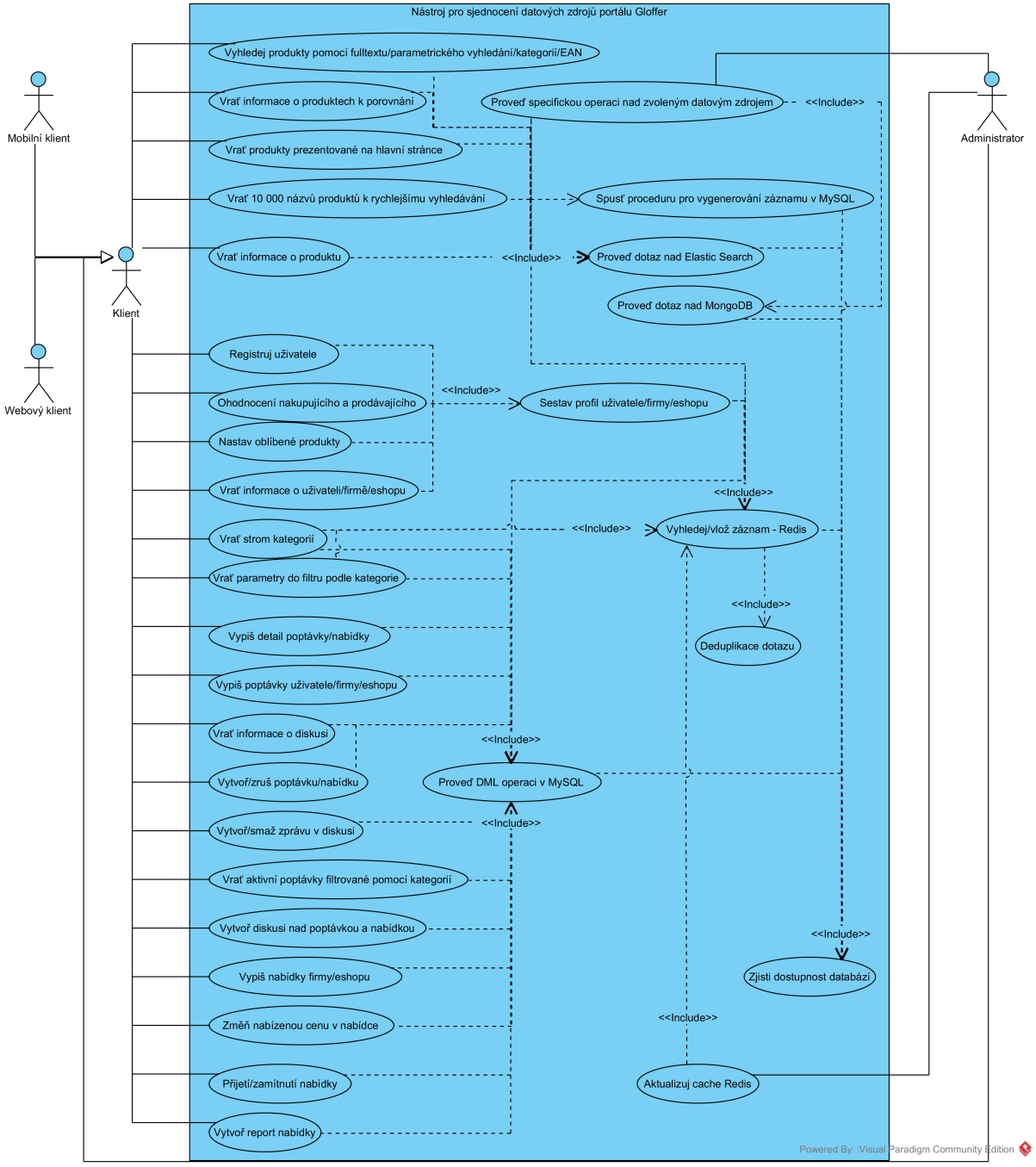
  Z této databáze pouze vracíme záznamy. Vkládání, aktualizaci a mazání opět zajišťuje jiná aplikace portálu Gloffer.

- **MySQL**: SELECT, UPDATE, INSERT, DELETE

  Práce s touto klasickou relační databází by měla být kompletní ve smyslu DML.

- **Redis**: SELECT, UPDATE, INSERT, DELETE

  V této cachovací databázi je možné zapisovat, aktualizovat, mazat i vracet záznamy.



Obrázek 15: Diagram případu užití navrhovaného systému

4.3.1 REST API Endpointy

Pro vytvoření funkcionality systému bylo potřeba definovat všechny endpointy, které v rámci aplikace budu muset zpracovávat.

Společné vlastnosti:

- *page* slouží ke stránkování produktů (vždy 20 výsledků na stránku) a indexace začíná od 0
- *type* specifikuje vlastnosti navráceného produktu
  - **hint**: ID produktu, titulek (cs, sk, en)
  - **simle**: *hint* + cena, měna, zdroj produktu, pole URL adres obrázků
  - **full**: *simple* + popis, vlastnosti produktu, nabídky firem a e-shopů, ID kategorie a informace o značce produktu
  - **huge**: *full* + obsah, EAN, statistiky

Všechny endpointy datového rozhraní jsou následující:

- **GET**
  - */product/{id}/{type}* - vrátí produkt s ID = **id**
  - */product/ean/{type}/{ean}* - vyhledá produkt s EAN = **ean**
  - */products/{id}/{type}/{page}* - vrátí produkty v kategorii s ID = **iD**
    Součástí endpointu mohou být také parametry pro řazení výsledků (podle ceny, názvu) a parametry pro filtrování produktů (parametrické vyhledávání). Ukázka předávání parametrů pro účely parametrického vyhledávání a třídění viz 14.
  - */products/{type}/{ids}* - vrátí produkty, jejichž ID ∈ **ids** (odděleny znakem '&')
  - */products/homepage* - vrátí produkty prezentované v mobilní aplikaci na hlavní straně
    Součástí endpointu mohou být také parametry pro řazení výsledků (podle ceny, názvu) a parametry pro filtrování produktů (parametrické vyhledávání). Ukázka předávání parametrů pro účely parametrického vyhledávání a třídění viz 14.
  - */products/all* - vrátí 10 000 názvů a ID produktů pro mobilní vyhledávač
  - */products/{id}/{type}/{fulltext}/{page}* - vyhledá produkty v kategorii s ID = **id** a odpovídající fulltextu **fulltext**
    Součástí endpointu mohou být také parametry pro řazení výsledků (podle ceny, názvu) a parametry pro filtrování produktů (parametrické vyhledávání). Ukázka předávání parametrů pro účely parametrického vyhledávání a třídění viz 14.
  - */categories/homepage* - vrátí názvy a ID kategorií v hlavní úrovni kategorií
  - */categories/{id}* - vrátí potomky (názvy a ID kategorií) kategorie s ID = **id**

- */user/{id}* - vrátí profilové informace o uživateli s ID = **id** viz 15
- */firm/{id}* - vrátí profilové informace o firmě s ID = **id** viz 17
- */shop/{id}* - vrátí profilové informace o e-shopu s ID = **id** viz 16
- */discussion/{id}* - vrátí diskusi s ID = **id**
- */requests/{type}/{id}* - vrátí poptávky pro daného uživatele, firmu nebo e-shop (specifikováno atributem **type**, který může být „user“, „firm“ nebo „shop“), kde ID (uživatele, firmy, e-shopu) = **id**
- */requests/{id}* - vrátí aktivní poptávky uživatelů, firem a e-shopů vyfiltrované podle kategorie, kde ID kategorie = **id** (vrací i poptávky produktů v podkategoriích)
- */responses/{type}/{id}* - vrátí nabídky pro danou firmu nebo e-shop (specifikováno atributem **type**, který může být „firm“ nebo „shop“), kde ID (firmy, e-shopu) = **id**
- */request/{id}* - vrátí poptávku s ID = **id**
- */response/{id}* - vrátí nabídku s ID = **id**
- */responseReport/{id}* - vrátí JSON report uzavřené nabídky s ID = **id**
- */filter/{id}* - vrátí parametry do filtru pro kategorii s ID = **id** (vrátí i parametry pro podkategorie)

- **POST**
  - */user* - registruje uživatele jehož vlastnosti jsou v těle JSON požadavku viz 19
  - */request* - vytvoří poptávku na produkt, vlastnosti poptávky jsou v těle JSON požadavku viz 32
  - */response* - vytvoří nabídku určitou poptávku, vlastnosti nabídky jsou v těle JSON požadavku viz 33
  - */discussion* - vytvoří diskusi k poptávce, vlastnosti diskuse jsou v těle JSON požadavku viz 34
  - */{db}* - nad zvolenou databází **db** provede dotaz **query** a vrátí výsledky v JSON formátu

    V těle JSON požadavku je pod klíčem **query** uložen příkaz k vykonání nad zvoleným datovým zdrojem.
    - **db** může nabývat hodnot
      - *mysql* - dotaz spouštěn nad MySQL
      - *mongodb* - dotaz spouštěn nad MongoDB
      - *elastic* - dotaz spouštěn nad Elastic Search
      - *redis* - dotaz spouštěn nad Redisem
    - **query** může být zadáno následovně

- *mysql* - SQL příkaz spustitelný nad MySQL viz 21
- *mongodb* - JSON obsahující klíče *database, find*, kde **database** je název databáze určené pro vykonání dotazu uloženého ve **find** (JSON dokument, který slouží jako argument pro vyhledávání) viz 20
- *elastic* - JSON obsahující klíče *method, endpoint, command*, kde **method** může nabývat hodnot (*GET, PUT, POST, DELETE*), **endpoint** obsahuje cílový datový zdroj pro provedení Elastic Search příkazu, který je uložen v JSON formátu u klíče **command** viz 23
- *redis* - JSON viz 22 obsahující klíče *type, key, value, ttl*, kde **type** může nabývat hodnot (*insert, delete, select*), **key** udává klíč vkládaného/mazaného/dotazovaného objektu a **value** se používá pouze u vkládání a obsahuje data v JSON formátu podle specifikace ukládání dat do Redisu viz 4.5 a **ttl** se používá pouze pro vkládání a specifikuje délku životnosti záznamu v minutách

- **PUT**
  - */refresh* - provede aktualizaci cache databáze Redis pro data popsaná v JSON požadavku viz 18
  - */favorite/{id}* - nastaví uživatelovu oblíbenost produktu s ID = **id** viz 24
  - */discussion/{id}* - přidá zprávu do diskuse s ID = **id**, vlastnosti zprávy jsou v těle JSON požadavku viz 36
  - */response/{id}* - upraví nabídku, kde upravované parametry jsou v těle JSON požadavku viz 35
  - */evaluate/{type}/{responseId}* - provede hodnocení nakupujícího nebo prodávajícího (specifikováno parametrem **type**, který může nabývat hodnot „buyer“ nebo „seller“) v nabídce s ID = **responseId**, parametry hodnocení jsou v těle JSON požadavku viz 37
  - */response/{id}/accept* - akceptování nabídky s ID = *id*
  - */response/{id}/refuse* - zamítnutí nabídky s ID = *id*
- **DELETE**
  - */request/{id}* - zruší poptávku s ID = **id**
  - */response/{id}* - zruší nabídku s ID = **id**
  - */message/{id}* - smaže zprávu z diskuse, ID zprávy = **id**

4.4 Návrh rozhraní ve vztahu k GDPR

General Data Protection Regulation neboli GDPR je obecné nařízení o ochraně osobních údajů vycházející z legislativy EU, která si klade za cíl zvýšit ochranu osobních dat občanů. K přijetí GDPR došlo v dubnu 2016, ale účinnosti nabyde 25. května 2018. Cílem tohoto nařízení je sjednocení pravidel týkajících se ochrany osobních údajů ve všech státech Evropské unie (a Islandu, Norska a Lichtenštějnska) [20].

Tímto obecným nařízením se bude muset zabývat a řídit každý subjekt, který zpracovává osobní údaje (jméno, příjmení, adresa, email atd.), tudíž i portál Gloffer, který ukládá informace o svých uživatelích a jejích prováděných akcích v rámci aplikace.

V případě nedodržení legislativních nařízení vyplývajících z GDPR je maximální výše pokut 20 000 000 €.

4.4.1 Problémy související s GDPR a jejich možné řešení

V rámci implementace funkcionality systému Gloffer bylo zapotřebí vyřešit otázky týkající se ochrany osobních údajů vyplývající z GDPR. Hlavní problémy jsou popsány v následujících kapitolách 4.4.1.1 až 4.4.1.4 a je nastíněno jejich řešení.

4.4.1.1 Prokazatelnost aktivit na straně správce a zpracovatele osobních údajů

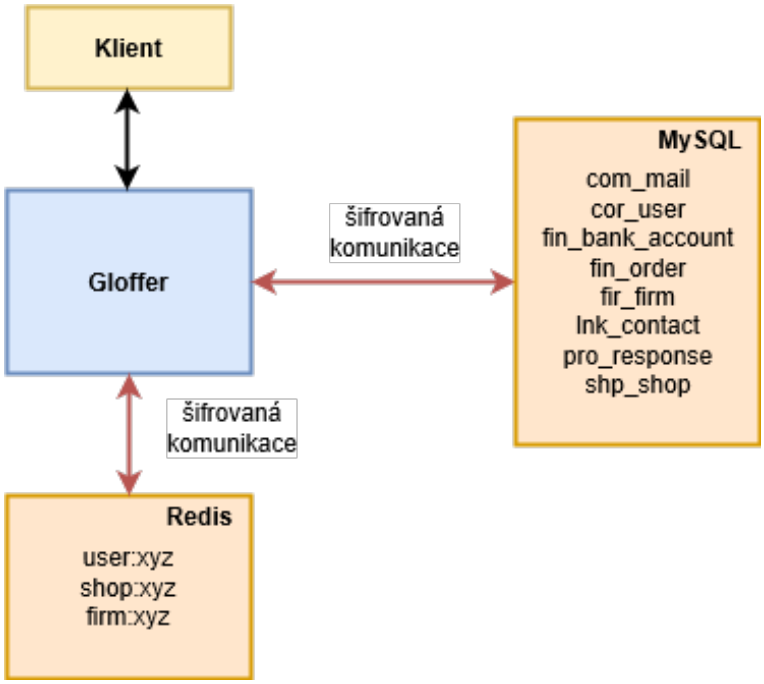
Správce a zpracovatel osobních údajů (v našem případě portál Gloffer) musí být schopen kontrolním orgánům prokázat aktivity spojené s osobními údaji (registrace, přihlášení, zpracování, poskytování uživatelských databází atd.).

Řešení: Systém pro sběr a ukládání informací o provozu portálu, tzv. logování. Každá akce provedená portálem, by měla být zaznamenávána do perzistentního datového úložiště (RDB, .log soubory atd.). Tyto záznamy by měly pouze vznikat a nikoliv zanikat (smazání, přepsání). Logovací systém by měl vykonávat pravidelné zálohy a nejlépe vytvářet redundantní kopii na vzdáleném úložišti.

4.4.1.2 Ochrana dat

V systému/podniku musí být zabezpečení dat před neoprávněným přístupem a nezákonným zpracováním. Jelikož bezpečnost dat je kritické místo každého systému a nové hrozby jsou na denním pořádku, je třeba zajistit aktuální potřeby ochrany.

Řešení: Šifrování dat ukládaných v cache Redis a relační databázi MySQL, kde se nachází informace o uživatelích, firmách a e-shopech, je hlavním řešením tohoto problému společně se zabezpečenou komunikací mezi uživatelem a systémem. Samotná komunikace systému s těmito dvěmi datovými zdroji by měla být šifrována. Schéma na obrázku 16 určuje, které data musí být šifrována a přenášena zabezpečeným přenosem.



Obrázek 16: Schéma šifrované komunikace mezi datovými zdroji a systémem

Jako další řešení se může jevit bezpečnost jako outsourcingová [11] služba, kdy specializované společnosti zajistí aktivní ochranu našich dat a náš podnik se může soustředit na hlavní cíle související s vývojem vlastního produktu.

4.4.1.3 Minimalizace potřebných osobních údajů

Další povinností je sběr informací pouze omezeného rozsahu v tom smyslu, že nebudeme moci požadovat po uživatelích osobní informace, které jsou k funkcionalitě našeho systému/podniku potřeba.

Řešení: Možným řešením je komplexnost našeho systému, kdy potřebuje jak kontaktní údaje uživatelů, tak adresy pro doručení, čísla karet pro bankovní transfery atd. Pokud pro správnou funkčnost našeho systému je věkové omezení pro uzavírání obchodu, tak je potřeba od uživatele získat věk (např. z rodného čísla).

4.4.1.4 Vymazání osobních údajů

Uživatel nebo zákazník si bude moci vyžádat výmaz svých osobních informací ze systému. Avšak k výmazu osobních údajů z databáze může dojít pouze pokud osobní údaje již nejsou potřebné pro účel, pro který byly shromážděny.

Řešení: Jako řešením se může jevit chytrá smlouva s uživatelem, kdy vytvoříme podmínky, za kterých v podstatě uživatel souhlasí s uchováním osobních informací „navěky“.

[11] Poskytování podpůrných služeb a procesů podniku nebo organizaci jiným podnikem nebo organizací.

4.5 Konvence ukládání do cache databáze Redis

Součástí diplomové práce bylo nastudovat problematiku ukládání předpřipravených dat do cache databáze Redis 3.8 a vytvořit pravidla pro vkládání záznamů do této databáze a nastavení jejich životnosti:

- Řetězce klíče vytváříme ve tvaru A:B:...:Z, kde A má nejnižší stupeň granularity dat a Z má nejvyšší stupeň granularity dat.
- První hodnotou (A) řetězce klíče je název objektu (*user, firm, shop, product, products, category, search, categoryProperties, reference*).
- Druhou hodnotou (B) řetězce klíče je jedinečný identifikátor objektu, ke kterému se předpřipravené data vztahují (*user ID, shop ID, firm ID, product ID, category ID, search hash*), nebo označení skupiny objektů (*all, homepage, result, ean*).
- Třetí hodnota (C) řetězce klíče se využívá především u objektů *search, products, product, reference, result*. V této hodnotě je vypočítán hash parametrů nacházejících se v URL (třídění a filtrování), nebo označení typu produktu (hint, simple, full, huge), nebo hash výsledku vyhledávání v případě *search:result*.
- Čtvrtá hodnota (D) a pátá hodnota (E) řetězce klíče je využívána u objektů *products a product* pro označení stránky výsledků, nebo kód EAN (D) a pro hash parametrů (E) nacházejících se v URL (třídění a filtrování) - E pouze pro *products*.
- TTL nastavíme podle typu objektu
  - *user, firm, shop* - nastavíme hodnotu na -1, což je neomezená platnost záznamu

    Při změně údajů uživatele, firmy nebo e-shopu v aplikaci Gloffer by mělo dojít k vygenerování nového JSON dokumentu a poté by měl být tímto dokumentem nahrazen předešlý záznam v Redisu.
  - *product* a *products*
    - *products:all* - nastavíme hodnotu na 259 200, což jsou 3 dny (slouží pouze k rychlejšímu vyhledání produktů)
    - *products* a *product* - nastavíme hodnotu na 3 600, což je 1 hodina

    Pokud TTL záznamu je stále aktivní a změní se produkt v Elastic Search, mělo by dojít k vygenerování nového JSON dokumentu a měl by být tímto dokumentem nahrazen předešlý záznam v Redisu.
  - *category* - nastavíme hodnotu na 259 200, což představuje životnost záznamu 3 dny

    Jelikož kategorie vznikají nebo se mění jejich názvy velice zřídka, není potřeba záznamy v cache paměti aktualizovat tak často, jako produkty nebo výsledky vyhledávání.

  - *search* - nastavíme hodnotu na 1800, což představuje životnost záznamu 30 minut

    Výsledky dotazů se mohou měnit nejčastěji, jelikož mohou vracet záznamy z více kategorií a tudíž je pravděpodobnost změny vyšší. Je potřeba tedy TTL nastavit tak, aby se změny včas promítly do výsledků dotazů, ale aby nedocházelo ke zbytečné zátěži díky znovu-generování stejného záznamu do Redisu.
  - *search:result* - hodnotu nastavíme na 604 800, což představuje životnost záznamu 7 dní (po tuto dobu jsou výsledky vyhledávání uloženy v Redisu - je to i z důvodu možné zpětné změny, kdy výsledek vyhledání opět nalezneme v Redisu)
  - *categoryProperties* - nastavíme hodnotu na 604 800, což představuje životnost záznamu 7 dní

    Vlastnosti kategorií většinou nevznikají nebo se nemění tak často, jako se mění produkty.
  - *reference* - TTL této skupiny je nastaveno na 604 800, což představuje životnost záznamu 7 dní

Informace ukládané pro jednotlivé objekty:

- *user:xyz* - Pro uživatele s ID = xyz ukládáme viz 15
- *firm:xyz* - Pro firmu s ID = xyz ukládáme viz 17
- *shop:xyz* - Pro e-shop s ID = xyz ukládáme viz 16
- *product:xyz:type* - Pro produkt s ID = xyz, kde **type**
  - *hint* - viz 25
  - *simple* - viz 26
  - *full* - viz 27
  - *huge* - viz 28
- *products:all* - ukládáme pole 10 000 produktů typu *hint*
- *products:homepage* - ukládáme pole 20 produktů typu *simple*
- *category:xyz* nebo *category:homepage* - pro kategorii s ID = xyz nebo ID kořenové kategorie ukládáme viz 29
- *categoryProperties:xyz* - pro kategorii s ID = xyz ukládáme parametry pro filtrování (např. výrobce) viz 30
- *reference:xyz* - pro odkazující záznam (pro možnosti aktualizace cache) s hashem odkazovaného dotazu = xyz ukládáme URL cestu pro opětovné vygenerování stejných výsledků a parametry z URL viz 31

- *search:uvw:xyz* - pro výsledek vyhledávání s hashem dotazu = uvw a hashem URL parametrů = xyz ukládáme hash výsledku vyhledávání (pomocí tohoto hashe poté přistupujeme k výsledku uloženém v *search:result*)

- *search:result:xyz* - pro výsledek vyhledávání s hashem výsledku vyhledávání = xyz ukládáme JSON dokument vyhledaných produktů

4.6 Problém párování dokumentů

Obrovským problémem, kterým se všichni členové vývojového týmu zabývali, byl problém párování dokumentů. Bylo potřeba zajistit, že se dokumenty spárují správně nejlépe podle nějakého unikátního klíče. Tímto klíčem by u produktu mohl být EAN, avšak tyto jedinečné číselné označení položek nebyly k dispozici.

Bylo třeba vymyslet, jak data spárovat tak, aby byla co největší míra úspěchu spárování produktů a ručně se musela dopárovávat jen malá část produktů.

4.6.1 Řešení v Elastic Search

Elastic Search nabízí řadu modelů pro vyhodnocení podobnosti dokumentů. Lze tedy využít těchto modelů k nalezení stejných nebo alespoň podobných dokumentů. Nejpoužívanější modely v Elastic Search jsou [21]:

- **BM25**

  Jedná se o model podobnosti, který je založen na pravděpodobnostním modelu, který odhaduje pravděpodobnost nalézaných dokumentů pro zadaný dotaz. Je založen na TF/IDF podobnosti, která pracuje dobře s malými dokumenty, kde se termy opakují.

  - TF - čím více se term dotazu objevuje v dokumentu, tím je pravděpodobnější, že dokument bude podobný.
  - IDF - čím více se term objevuje v dokumentech v kolekci dokumentů, tím se snižuje skóre. Termy, které nejsou tolik obvyklé, jako např. „elasticsearch“, nám pak pomáhají zpřesnit výsledky.

- **DFR**

  Model podobnosti je založen na pravděpodobnostním modelu stejného jména a je výhodné jej použít na podobnost textů v přirozeném jazyce.

- **DFI**

  Model podobnosti, který implementuje DFI model, který je založen na Chi-kvadrát [12] statistice.

[12] Spojité rozdělení pravděpodobnosti, které má význam pro určení, zda množina dat vyhovuje dané distribuční funkci.

- **IB**

  Model je založen na konceptu, že informační obsah v sekvenci symbolů je primárně určen opakujícím použitím jeho základních elementů. Je velice podobný DFR modelu a tudíž je výhodný pro vyhledání podobnosti textů v přirozeném jazyce.

- **LM Dirichlet**

  Model podobnosti využívající Bayesovské vyhlazování s využitím Dirichletových předků.

- **LM Jelinek Mercer**

  Jedná se o model podobnosti, který se pokouší zachytit důležité vzory v textu. Je založen na Jelínkových-Mercerových vyhlazovacích metodách.

Více se o modelech podobnosti můžeme dozvědět na `https://www.elastic.co/guide/en/elasticsearch/reference/5.6/index-modules-similarity.html`.

5 Implementace

Proces samotné implementace spočíval v naprogramování funkcionality popsané v 4. Jako zvolený programovací jazyk byla zvolena Java ve verzi 1.8 s využitím Spring Frameworku ve verzi 5.0.5 a jako vývojové prostředí bylo zvoleno IntelliJ IDEA.

5.1 Maven - přidávání závislostí

Maven je nástroj pro usnadnění sestavování aplikací a správu projektů vytvořených v programovacích jazycích založených na Javě. V našem projektu slouží především k přidávání externích knihoven do projektu.

V projektu pracujeme s MySQL, Redis, MongoDB a Elastic Search databázemi, tudíž bylo potřeba přidat závislost projektu na API klientů těchto technologií. Tuto funkcionalitu poskytuje externí knihovna *cz.gloffer.backend*, kterou jsem přidal do *pom.xml* viz 4. Zajišťuje jak práci s databázemi, tak poskytuje třídy pro práci s výsledky získanými z Elastic Search datového zdroje.

```
<dependency>
   <groupId>cz.gloffer.backend</groupId>
   <artifactId>core</artifactId>
   <version>LATEST</version>
</dependency>
```

Výpis 4: Přidání externí knihovny pro práci s databázemi přes Maven (pom.xml)

Dalšími důležitými využívanými externími knihovnami v navrženém systému jsou:

- org.json: json
- org.springframework.boot:
  - spring-boot-starter-jdbc
  - spring-boot-starter-data-jpa
  - spring-boot-starter-test
  - spring-boot-starter-tomcat
- org.springframework.data: spring-data-redis
- redis.clients: jedis
- net.openhft: zero-allocation-hashing
- mysql: mysql-connector-java
- com.fasterxml.jackson.core: jackson-databind

5.2 Konfigurace

Třída **AppConfig** označená jako *@Configuration* značí, že tato třída slouží jako konfigurační. Tato třída slouží pro definování služeb **DataService, UserDataService, FirmDataService, ShopDataService, CategoryDataService, ProductDataService, RequestDataService**, které jsou označeny jako *@Bean* tedy jako komponenty systému, díky čemuž mohou být tyto komponenta dále v systému používány.

Další součástí této třídy jsou statické metody vracející konfigurační údaje pro připojení k databázím. Definují také URL cestu k obrázkům, předpony klíčů, endpointy Elastic Search databáze, ale také TTL pro nastavení cache databáze. V neposlední řadě poskytují metody pro zajištění funkcionality dostupnosti služeb (resetování čítačů nedostupnosti, časovač a další).

Další konfigurační třídou je **RedisConfig** viz 5, která slouží pro vytvoření spojení s cache Redis. Poskytuje schéma pro přístup k záznamům Redisu.

```
@Configuration
public class RedisConfig {
    @Bean
    JedisConnectionFactory jedisConnectionFactory() {
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
        jedisPoolConfig.setMaxWaitMillis(120000);
        JedisConnectionFactory factory = new JedisConnectionFactory();
        factory.setHostName(AppConfig.getRedisHost());
        factory.setPort(AppConfig.getRedisPort());
        factory.setPassword(AppConfig.getRedisPassword());
        factory.setPoolConfig(jedisPoolConfig);
        factory.setTimeout(120000);
        factory.setUsePool(true);
        return factory;
    }

    @Bean
    RedisTemplate<String, Object> redisTemplate() {
        final RedisTemplate<String, Object> template = new RedisTemplate<String
            , Object>();
        template.setConnectionFactory(jedisConnectionFactory());
        template.setKeySerializer(new StringRedisSerializer());
        template.setHashValueSerializer(new GenericToStringSerializer<Object>(
            Object.class));
        template.setValueSerializer(new GenericToStringSerializer<Object>(
            Object.class));
```

```
        template.afterPropertiesSet();
        return template;
    }
}
```

Výpis 5: Konfigurační třída pro práci s Redisem

5.3 Databázové konektory

Databázové připojení ke všem databázovým technologiím (MySQL, Elastic Search, Redis, MongoDB) je pro toto rozhraní klíčové. Hlavní datová služba **DataServiceImpl** proto ustavuje spojení s výše zmíněnými datovými zdroji. MySQL, MongoDB a Elastic Search jsou vytvořeny jako statické proměnné. Redis šablona je automaticky nastavena pomocí notace *@Autowired* (není potřeba operací pokrývající nastavení a vrácení proměnné).

Jednotlivé parametry pro nastavení připojení k datovým zdrojům je součástí konfigurační třídy AppConfig viz 5.2, ve které jsou tyto parametry přístupné prostřednictvím veřejných metod.

```
private static MongoDBConnector mongo;
private static ElasticsearchLowLevelConnector elastic;
private static Connection mysql;

@Autowired
private RedisTemplate<String, Object> template;

static
{
   AppConfig.resetAllTry();
   MongoDBConfig mongoDBConfig = new MongoDBConfig();
   ArrayList<String> mongoHosts = new ArrayList<String>();
   mongoHosts.add(AppConfig.getMongoHost());
   mongoDBConfig.addServerAddresses(mongoHosts);
   mongo = new MongoDBConnector(mongoDBConfig);

   ElasticsearchConfig elasticSearchConfig = new ElasticsearchConfig();
   ArrayList<String> elasticSearchHosts = new ArrayList<String>();
   elasticSearchHosts.add(AppConfig.getElasticSearchHost());
   elasticSearchConfig.addHttpHosts(elasticSearchHosts);
   elastic = new ElasticsearchLowLevelConnector(elasticSearchConfig);
```

```
    try {
        mysql = DriverManager.getConnection(
            "jdbc:mysql://" + AppConfig.getMysqlHost() + "/gloffer",
            AppConfig.getMysqlUsername(),
            AppConfig.getMysqlPassword()
        );
    } catch (SQLException e) {
        AppConfig.incMysqlTry();
    }
}
```

Výpis 6: Statický blok kódu pro vytvoření databázových spojení

5.4 Objekty

Pro naši funkcionalitu vzniklo v rozhraní několik tříd pro serializaci objektů. Jsou to tyto třídy:

- **Category** - kategorie
- **CommonData** - společné vlastnosti pro objekty třídy UserData, FirmData a ShopData
- **Discussion** - diskuse
- **FirmData** - firma
- **Message** - zpráva v diskusi
- **UserData** - uživatel
- **ShopData** - e-shop
- **ProductHint** - produkt
- **ProductSimple** - rozšiřuje ProductHint
- **ProductFull** - rozšiřuje ProductSimple
- **ProductHuge** - rozšiřuje ProductFull
- **Request** - poptávka
- **Response** - nabídka
- **ResponseData** - report nabídky

Všechny tyto třídy mají své třídní diagramy v rámci elektronické přílohy v adresáři *Images*.

5.5 Kontrolér

Kontrolér **DataRestController** vznikl pro obsluhu REST požadavků přicházejících z mobilní, webové, ale i administrátorské desktopové aplikace. V této třídě tedy najdeme všechny možné endpointy viz 4.3.1, které měly být v rámci diplomové práce implementovány. Je zde využito notace Spring frameworku, který nám provádí mapování těchto požadavků a vytvoření odpovědi na tyto požadavky. Ukázkovou implementaci endpointu s využitím notace Spring frameworku lze vidět v kódu viz 7.

```
@RequestMapping(value = "/{db}", method = RequestMethod.POST)
@ResponseBody
public String performQuery(
   @PathVariable("db") String db,
   @RequestBody String body
) {
   return dataService.performQuery(db, body);
}
```

Výpis 7: Ukázka implementace endpointu pomocí Spring frameworku

- **@RequestMapping** - specifikuje mapování požadavku přicházejícího na server
- **@ResponseBody** - definuje tělo odpovědi a způsob jakým ho získat.

5.6 Služby

Služby pokrývající funkcionalitu systému a zpracovávající data pro výsledky požadavků byly rozděleny do několika tříd a k nim odpovídajícím rozhraním podle toho, nad kterými objekty pracují. Bylo vytvořeno celkem 7 služeb:

5.6.1 CategoryDataService

Tato služba je zodpovědná za práci s kategoriemi. Poskytuje metody pro vrácení informací o podkategoriích zvolené kategorie. Využívá ke své funkcionalitě MySQL a Redis připojení a pro serializaci objektů využívá třídy Category.

5.6.2 DataService

DataService je hlavní službou pokrývající logiku vykonání specifického dotazu nad konkrétním datovým zdrojem. Dále poskytuje operační logiku pro cache Redis. Pro svou funkcionalitu využívá MySQL, MongoDB, Elastic Search a Redis připojení. Zde se neprovádí žádná serializace do objektu konkrétní třídy.

5.6.3 FirmDataService

Služba zodpovědná za práci s firmami. Poskytuje metody pro vrácení informací o firmě a pro sestavení objektu třídy FirmData, kterou využívá k serializaci. Vyžaduje MySQL a Redis připojení.

5.6.4 ProductDataService

ProductDataService je nejkomplexnější implementovanou službou starající se o logiku produktů. Poskytuje metody pro vyhledání produktů v různých datových zdrojích a podle různých kritérií (parametry, fulltext, ID, EAN, kategorie). Dále poskytuje metodu pro aktualizaci záznamů v cache Redis, metodu vracející parametry filtrů zvolené kategorie a metodu vracející všechny podkategorie sloužící pro vyhledávání v MySQL. Pro svou funkcionalitu využívá MySQL, MongoDB, Elastic Search a Redis připojení. Pro serializaci objektů se využívá tříd ProductHint, ProductSimple, ProductFull a ProductHuge podle zvoleného typu výpisu informací o produktech.

5.6.5 RequestDataService

RequestDataService obstarává logiku poptávek a nabídek. Poskytuje tedy metody pro vytvoření nabídky, poptávky, diskuse a její zprávy. Dále nabízí metody pro zobrazení poptávek uživatele/e-shopu/firmy, nabídek e-shopu/firmy, zrušení nabídky a poptávky, smazání zprávy v diskusi, vytvoření JSON reportu po uzavření obchodu, přijmutí a zamítnutí nabídky, zobrazení detailu poptávky a nabídky, aktualizaci ceny nabídky, zobrazení aktivních poptávek podle filtru a ohodnocení uživatele/e-shopu/firmy. Pro svou funkcionalitu využívá pouze MySQL připojení. Pro serializaci objektů využívá tříd Discussion, Message, Request, Response, ResponseData.

5.6.6 ShopDataService

Služba zodpovědná za práci s e-shopy. Poskytuje metody pro vrácení informací o e-shopu a pro sestavení objektu třídy ShopData, kterou využívá k serializaci. Vyžaduje MySQL a Redis připojení.

5.6.7 UserDataService

Služba zodpovědná za práci s uživateli. Poskytuje metody pro vrácení informací o uživateli, nastavení oblíbenosti produktů, registraci uživatele a pro sestavení objektu třídy UserData, kterou využívá k serializaci. Vyžaduje MySQL a Redis připojení.

5.7 Logování databázových operací

Součástí vývoje systému je také zpětný přehled o provedených akcích nad datovými zdroji. K tomuto účelu jsem v nezávislé MySQL databázi vytvořil tabulku viz 8 sloužící k logování všech operací nad datovými zdroji (Elastic Search, MySQL, Redis, MongoDB). Tyto záznamy poté mohou sloužit k analýze vytíženosti datových zdrojů a také k určení, které dotazy se opakují a mohly by být předpřipraveny a uloženy v cache databázi (Redis).

```
CREATE TABLE log_data_api (
   id INT NOT NULL AUTO_INCREMENT,
   type ENUM('elastic','mysql','mongo','redis') NOT NULL,
   created DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   request TEXT NOT NULL,
   body_request JSON NULL,
   response JSON NULL,
   method ENUM('get','insert','select','update','delete') NOT NULL,
   PRIMARY KEY (id)
)
```

Výpis 8: Tabulka pro ukládání logů rozhraní

K samotnému logování poté slouží asynchronní funkce *log(...)* viz 9, která provádí samotné vkládání logu do tabulky *heureka_test.log_data_api*. Díky spouštění v novém vlákně nedochází logováním k nechtěnému zpoždění navrácení dat REST požadavkům.

```
public void log(String type, String request, String body_request, String
    response, String method)
{
   Runnable task = () -> {
      try {
         try {
            PreparedStatement preparedStatement = logger.prepareStatement(
               "INSERT INTO log_data_api (
                  type,
                  request,
                  body_request,
                  response,
                  method
               ) VALUES (?, ?, ?, ?, ?)"
            );
            preparedStatement.setObject(1, type);
            preparedStatement.setObject(2, request);
```

```
            preparedStatement.setObject(3, body_request);
            preparedStatement.setObject(4, response);
            preparedStatement.setObject(5, method);
            preparedStatement.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
};
new Thread(task, "LoggingThread").start();
}
```

Výpis 9: Funkce pro logování operací nad datovými zdroji

5.8 Statistiky vyhledávaných frází fulltextu

Další vlastností systému je zaznamenávání četností vyhledávaných frází ve fulltextovém vyhledávání. Vznikají tak statistiky, které mohou být využity pro potřeby analýzy a sestavování výsledku vyhledávání před samotným zavoláním funkce pro získání těchto výsledků. Tím se lze účinně vyhnout zpoždění, které je způsobeno získáním odpovídajících dat z datových zdrojů. Pro tuto funkcionalitu vznikla v MySQL databázi tabulka viz 10, do které ukládáme hash vypočtený z hledané fráze (xxHash 5.12.1.1), hledanou frázi, datum prvního vytvoření záznamu, datum posledního vyhledání fráze a počet vyhledání.

```
CREATE TABLE log_fulltext_api (
    hash BIGINT NOT NULL,
    text VARCHAR(50) NOT NULL,
    created DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated DATETIME DEFAULT NULL,
    count INT NOT NULL DEFAULT 1,
    PRIMARY KEY (hash)
)
```

Výpis 10: Tabulka pro ukládání statistik vyhledávaných frází fulltextu

K aktualizaci statistik slouží asynchronní funkce *fulltextLog(Long hash, String fulltext)* viz 11. Tato funkce provádí vytvoření nového záznamu, pokud záznam s tímto klíčem neexistuje, jinak provede aktualizaci záznamu. Funkce se provádí v novém vlákně a tím nedochází k nechtěnému zpoždění.

```
public static void fulltextLog(Long hash, String fulltext) {
   Runnable task = () -> {
      try {
         try {
            Date currentDate = new Date();
            PreparedStatement preparedStatement = logger.prepareStatement(
               "INSERT INTO heureka_test.log_fulltext_api (
                  hash,
                  text
               ) VALUES (?,?)
               ON DUPLICATE KEY UPDATE
                  updated = ?,
                  count = count + 1"
            );
            preparedStatement.setObject(1, hash);
            preparedStatement.setObject(2, fulltext);
            preparedStatement.setObject(3,
               new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(currentDate)
            );
            preparedStatement.execute();
         } catch (SQLException e) {
            e.printStackTrace();
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   };
   new Thread(task, "LoggingFulltextThread").start();
}
```

Výpis 11: Funkce pro aktualizaci statistik fulltextového vyhledávání

5.9 Logika dostupnosti služeb a zpracování dat

Dostupnost služeb je řešena pomocí časovače při nedostupnosti služby, tzn. pokud služba při některém dotazu spadne do catch bloku znamenající nedostupnost této služby, dojde k nastavení časovače a v intervalu 3 minut není služba využívána. V tomto případě je dotazován alternativní datový zdroj, pokud je dostupný. Tato funkcionalita je součástí třídy *AppConfig* a její metody jsou využívány v rámci všech implementovaných služeb. Ukázka metod viz 12. Metoda **isMongoAvailable()** je poté volána před dotazy vyžadující MongoDB a pokud tato metoda vrátí ***true***, tak dojde k dotazování tohoto datového zdroje. Stejně tak i v případě ostatních datových zdrojů.

```
public static void startTimerMongo() {
   startTimeMongo = System.currentTimeMillis();
}

public static boolean isMongoAvailable() {
   if(maxTry <= getMongoTry()) {
      startTimerMongo();
      resetMongoTry();
      return false;
   }

   if(startTimeMongo + timeLimit < System.currentTimeMillis())
      return true;
   else
      return false;
}

public static int getMongoTry() {
   return mongoTry;
}

public static void incMongoTry() {
   mongoTry++;
}

public static void resetMongoTry() {
   mongoTry = 0;
}
```

Výpis 12: Metody pro zajištění logiky dostupnosti služeb pro MongoDB

V případě, že datový zdroj pro produkty vrátí ***null*** nebo prázdné pole produktů, dojde k hledání v dalším datovém zdroji až na nejnižší úroveň a tou je MySQL databáze. Tím je zajištěno 100% navrácení informací k produktům, pokud existují.

U fulltextového dotazování dochází k transformaci vyhledávané fráze na malé písmena, jelikož Elastic Search vyhledávač vrací stejné záznamy jak pro malé, velké i smíšené řetězce složené z písmen. Díky této transformaci dochází k šetření adresním prostorem v cache Redis a také zajišťuje správnou funkcionalitu statistik fulltextového vyhledávání a prodlužování životnosti záznamů v cache Redis.

5.10 Prodlužování životnosti záznamu v cache Redis

Pro implementaci funkcionality zajišťující prodlužování životnosti záznamu v cache Redis byla v MySQL databázi vytvořena tabulka *query_lifetime* viz 13. Do této tabulky ukládáme Redis klíč a hash výsledku.

Tři možné situace viz 17:

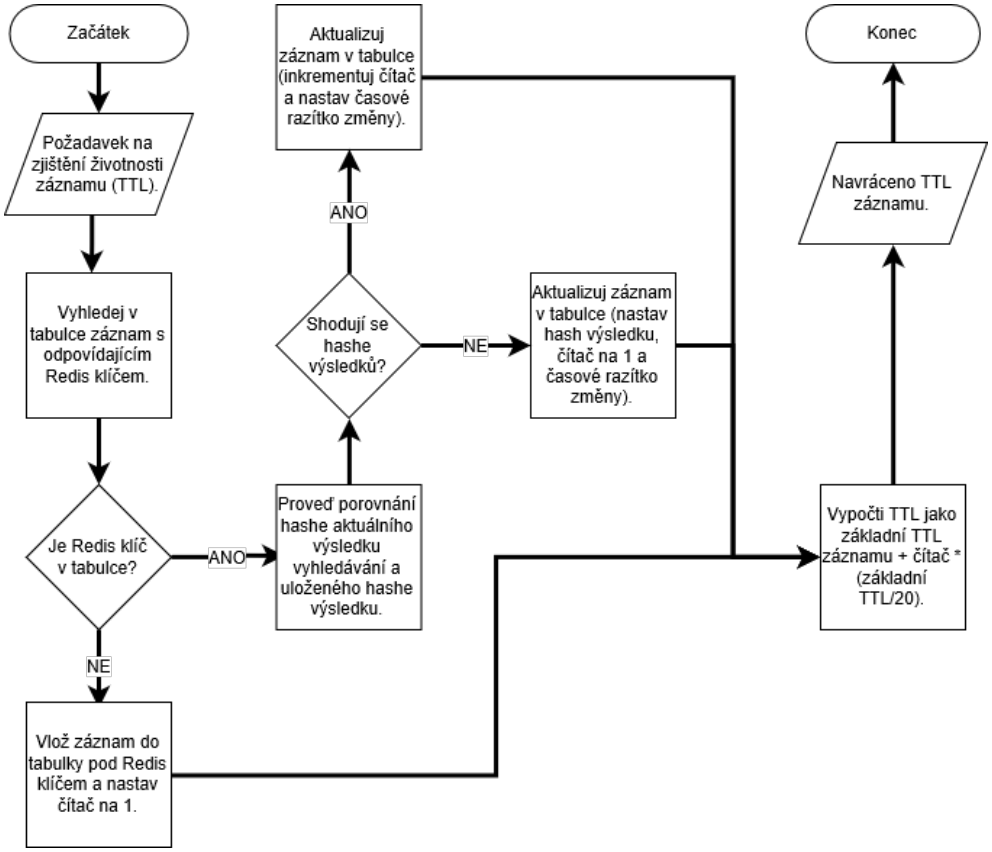
- V tabulce není odpovídající Redis klíč
  - Provede se vložení a nastavení čítače na hodnotu 1
- V tabulce je odpovídající Redis klíč a hashe výsledků se shodují (předešlý i aktuální)
  - Provede se aktualizace záznamu (čítač zvýší svou hodnotu o 1 a nastaví se časové razítko změny)
- V tabulce je odpovídající Redis klíč, ale hashe výsledků se neshodují (předešlý s aktuálním)
  - Provede se aktualizace záznamu (nastaví se hash aktuálního výsledku, čítač se nastaví na hodnotu 1 a nastaví se časové razítko změny)

Hodnota TTL se nastaví podle vzorce (**ttl** je standardní TTL vkládaného záznamu viz 4.5):

$$ttl + \text{čítač} \cdot \frac{ttl}{20}$$

```
CREATE TABLE query_lifetime (
  redis_key VARCHAR(80) NOT NULL,
  hash_result BIGINT(20) NOT NULL,
  count INT NOT NULL DEFAULT 1,
  created DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated DATETIME DEFAULT NULL,
  PRIMARY KEY (redis_key)
)
```

Výpis 13: Tabulka pro nastavování životnosti záznamu



Obrázek 17: Proces zjištění životnosti záznamu

5.11 Nástroj pro hromadnou aktualizací výsledků vyhledávání v cache Redis

Nástroj hromadné aktualizace záznamů v cache Redis slouží k rychlému opětovnému vygenerování stávajících záznamů daných produktů a uložení do cachovacího systému (aktualizace dat). Navíc tento nástroj dokáže neexistující záznamy do tohoto systému vygenerovat při dodržení konvence názvu klíče (tzn. sestaví produkt podle ID, podle EAN kódu, vytvoří seznam produktů pro nápovědu do mobilní aplikace). Nesmí se však jednat o výsledky vyhledávání podle fulltextu nebo kategorií, kde se vytváří hash z parametrů sloužících k třídění výsledku a filtrování.

Funkčnost tohoto nástroje je pro fulltextové dotazy a dotazy filtrující výsledky spočívá v tom, že při vytvoření záznamu do Redisu se současně vytvoří záznam pod klíčem *reference:**hash*** (*hash* je stejný jako u vytvářeného záznamu s daty o produktech), ve kterém je uložen REST endpoint pro vytvoření tohoto záznamu společně se seznamem parametrů pro vytvoření dotazu.

Při požadavku na aktualizaci záznamu je vyhledán klíč *reference:**hash*** a díky znalosti endpointu je provedena požadovaná operace pro vytvoření nového záznamu.

Vstupem pro tento nástroj jsou jednotlivé klíče Redisu (v těle JSON souboru pole s názvem *keys*) nebo celé domény obsahující sady klíčů (v těle JSON souboru pole s názvem *domains*). Z jednotlivých domén získáme všechny klíče a přidáme je k ostatním (klíče v *keys*. Z důvodu

nezasahování do cizího Redis úložiště je možné takto aktualizovat pouze výsledky vyhledávání produktů v rámci naší API. Hlavní část kódu zajišťující aktualizaci záznamů viz 38.

5.12 Deduplikace výsledků fulltextového vyhledání

Problémem u fulltextového vyhledávání je fakt, že pro různé vyhledávané fráze můžeme získávat stejné výsledky. To má negativní vliv na plýtvání místem v cachovací databázi Redis.

Řešení tohoto problému (celý proces fulltextového vyhledávání viz 18):

1. Pro fulltextový dotaz vytvořím hash endpointu (**hashQ**), hash URL parametrů předávaných v endpointu (**hashP**)
2. Do Redisu uložím záznam s endpointem a hodnotami URL parametrů pod klíčem *reference:**hashQ**:**hashP***
3. Provedu fulltextové vyhledání záznamu a vytvořím hash výsledku vyhledávání (**hashR**)
4. Do Redisu uložím záznam s **hashR** pod klíčem *search:**hashQ**:**hashP***
5. Do Redisu uložím záznam s výsledkem vyhledávání pod klíčem *search:result:**hashR***

Díky tomuto řešení nevznikají v Redisu duplicitní výsledky vyhledávání, které mohou plýtvat místem. Navíc toto řešení je rychlé, jelikož využívá pouze operací nad cache Redis, které mají konstantní časovou složitost. Součástí tohoto úkolu bylo taky zajistit, aby výpočet hashe malých i velkých dokumentů bylo co nejrychlejší. Bylo zapotřebí zvolit vhodnou hashovací funkci.

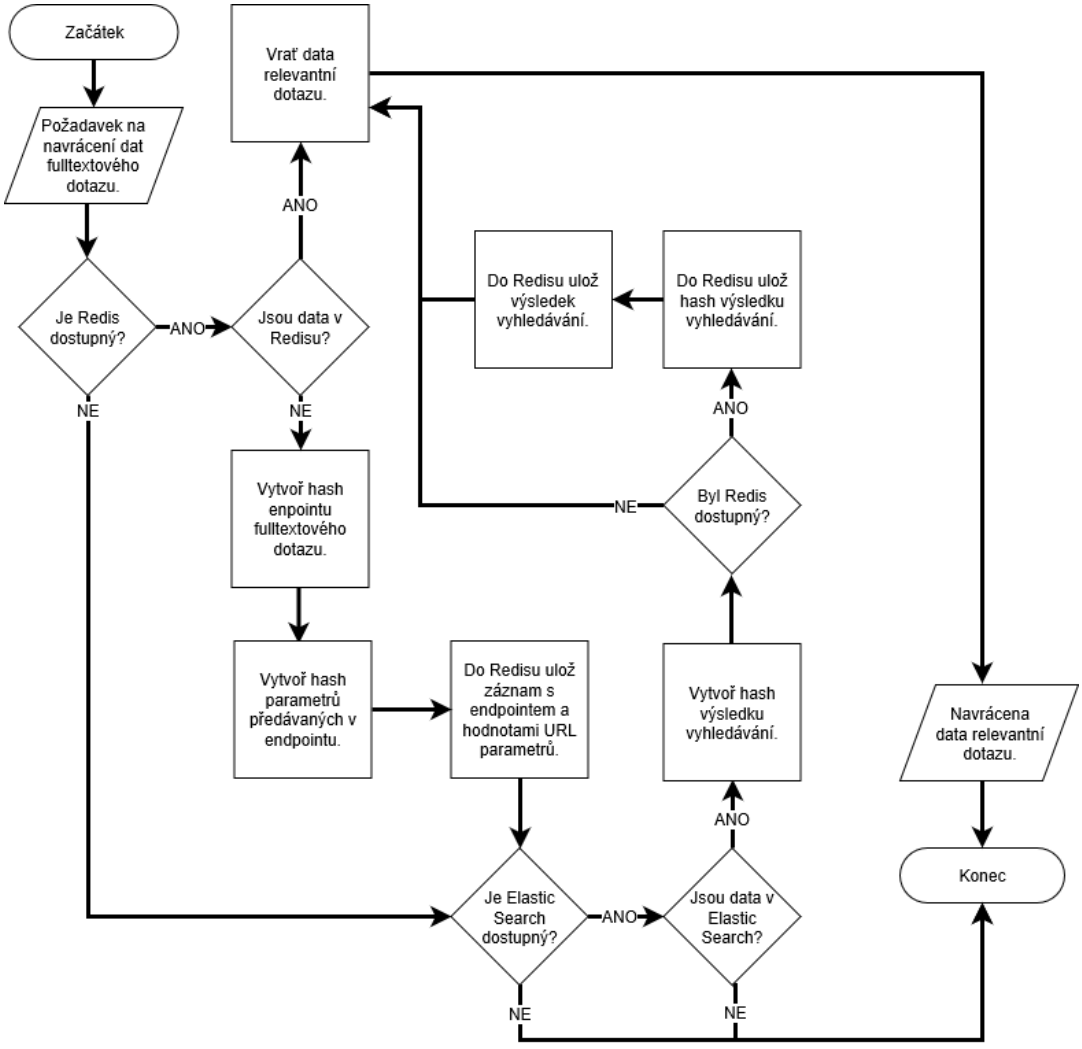
5.12.1 Hashovací algoritmus

Jelikož se hashe v naší aplikaci budou počítat často, tj. pro každý dotaz, tak je potřeba, aby hashovací funkce byla rychlá a pravděpodobnost kolize byla co nejmenší. Standardní známé hashovací algoritmy jako je SHA-x a MD5 neposkytují takový výkon jako v případě nešifrujících hashovacích algoritmů.

Součástí této diplomové práce byl proveden experiment nad dvěma JSON dokumenty. První dokument je průměrně dlouhý JSON požadavek představující tělo Elastic Search dotazu, druhý dokument představuje průměrně dlouhou JSON odpověď zpracovaných dat z naší aplikace. Oba dokumenty popisuje tabulka 4. Testování proběhlo v 10 000 iteracích a byl vypočítán průměrný čas každého algoritmu. Výsledky lze vidět v tabulce 5 a na grafu 19. Na základě tohoto testu byl algoritmus xxHash zvolen pro náš případ použití.

5.12.1.1 xxHash

Jedná se o rychlý nešifrující hashovací algoritmus, který je dostupný ve dvou verzích (32 a 64-bitový otisk). Využívá se v databázových systémech, jako je MySQL, PrestoDB, ArangoDB, dále pak ve vývoji počítačových her, v aplikacích pro správu souborů atd. Implementace xxHash algoritmu je dostupná pro většinu programovacích jazyků (Java, Javascript, C#, PHP atd.). [41]



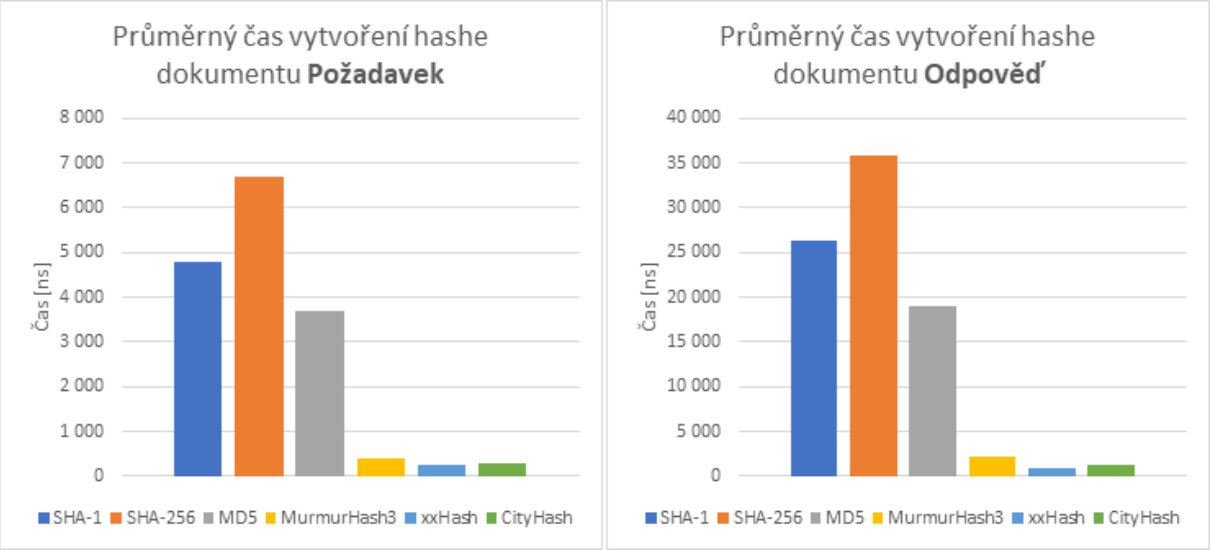
Obrázek 18: Proces deduplikace výsledků fulltextového vyhledávání

Tabulka 4: Vlastnosti testovacích dokumentů pro výběr hashovacího algoritmu

Dokument	Počet znaků	Počet řádků	Velikost [B]
JSON požadavek	761	37	796
JSON odpověď	4 843	183	4 910

Tabulka 5: Výsledky testů pro výběr hashovacího algoritmu po 10 000 iteracích

	Průměrný čas [ns]	
Algoritmus	Dokument **Požadavek**	Dokument **Odpověď**
SHA-1	4 798	26 382
SHA-256	6 689	35 865
MD5	3 691	18 955
MurmurHash3	397	2 176
xxHash	**236**	**919**
CityHash	277	1 290



Obrázek 19: Graf srovnání rychlostí hashovacích algoritmů

6 Testování

Součástí každého implementačního projektu je také fáze testování. Měli bychom zjistit, zda naše aplikace poskytuje dostatečný výkon pro nasazení v reálném provozu a zda funkcionalita odpovídá specifikaci projektu. Snažíme se objevit chybu na začátku projektu, kdy oprava této chyby není pro projekt tak nákladná jako po objevení chyby až v případě reálného provozu aplikace.

6.1 Testování funkcionality

Funkcionalita nástroje pro sjednocení datových zdrojů portálu Gloffer byla testována v průběhu jejího vývoje i po jejím dokončení. Všechny vytvořené endpointy tak byly zkontrolovány jak programátorem, tak uživatelem (tvůrcem mobilní aplikace). Případné nesrovnalosti byly ihned implementovány. Úkolem bylo ověřit, že navržená funkcionalita systému provádí přesně to, co se od ní očekává. Tato podmínka tedy byla splněna, jelikož navržený systém se chová očekávaným způsobem a poskytuje data, která jsou požadována.

6.2 Testování výkonu

Pro testování výkonu byly provedeny výkonové testy pomocí nástroje JMeter, ve kterém lze jednoduše nastavit scénáře pro testování. Testovaly se pouze operace vracející data, tzn. vytváření, aktualizace a mazání záznamů se netestovalo. Celkem byly provedeny čtyři testy:

1. 1 uživatel bez zpoždění při 200 opakováních bez uložených dat v cache Redis
2. 1 uživatel bez zpoždění při 200 opakováních s uloženými daty některých dotazů v cache Redis
3. 200 uživatelů se zpožděním 5 sekund při 1 opakování bez uložených dat v cache Redis
4. 200 uživatelů se zpožděním 5 sekund při 1 opakování s uloženými daty některých dotazů v cache Redis

První a druhý test má definovat průměrné časy odezvy na jednotlivé operace vracející nějaké záznamy. Třetí a čtvrtý test se pokouší otestovat reálné zatížení aplikace. Výsledky prvních dvou testů jsou součástí tabulky 6. Výsledky druhé dvojice testů jsou součástí tabulky 7. Výkon velmi závisí na propustnosti sítě a dostupném datovém zdroji. Testování navíc probíhalo přes VPN připojení na školní síť.

Provedené testy potvrdily užitečnost využití cache Redis. Průměrná odezva se snížila několikanásobně. Např. v případě získání detailu uživatele se jedná o 1/8 potřebného času. Pro optimální výkon celé aplikace by bylo vhodné mít v cache uloženo vše, co může být požadováno k vrácení více než jedenkrát. Avšak to není z důvodu velikosti dat možné. Byla proto nastavena logika ukládání záznamů do Redisu tak, aby výsledky vyhledávání měly určitou omezenou životnost.

Tabulka 6: Statistika výkonnostních testů pro 1 uživatele bez zpoždění při 200 opakováních

Před uložením dat do cache Redis				
	Čas [ms]			
Test	AVG	Min	Max	**Propustnost [op/min]**
CompareProducts	1 081	248	6 868	4,7
GetCategoriesByID	692	38	4 479	4,8
GetCategoriesHomepage	122	38	1 669	4,8
GetDiscussion	326	37	2 862	4,8
GetFilterParametersByCategory	1 543	189	82 206	4,5
GetFirmByID	333	38	2 205	4,8
GetProductByEAN	211	38	2 427	4,7
GetProductByID	1 191	205	9 174	4,7
GetProductsAll	2 226	581	29 826	4,7
GetProductsByCategory	1 207	168	6 839	4,7
GetProductsByFulltext	611	198	3 998	4,8
GetProductsHomepage	1 188	283	5 132	4,7
GetRequestByID	463	38	2 248	4,7
GetRequests	397	38	4 217	4,8
GetResponseByID	503	38	2 955	4,7
GetShopByID	328	38	3 094	4,8
GetUserByID	858	38	7 831	4,8
Celkem:	**781**	**37**	**82 206**	**80,5**

Po uložení dat do cache Redis				
	Čas [ms]			
Test	AVG	Min	Max	**Propustnost [op/min]**
CompareProducts	338	112	2 036	6,3
GetCategoriesByID	133	38	2 121	6,4
GetCategoriesHomepage	109	39	1 125	6,4
GetDiscussion	444	151	3 024	6,4
GetFilterParametersByCategory	271	38	13 411	6,2
GetFirmByID	118	38	1 026	6,4
GetProductByEAN	164	38	1 413	6,3
GetProductByID	119	38	936	6,3
GetProductsAll	2 419	588	15 666	6,3
GetProductsByCategory	1 410	386	11 342	6,3
GetProductsByFulltext	595	198	3 257	6,4
GetProductsHomepage	1 310	414	9 184	6,3
GetRequestByID	755	230	4 065	6,3
GetRequests	576	156	3 990	6,4
GetResponseByID	709	234	3 488	6,3
GetShopByID	90	38	687	6,4
GetUserByID	106	38	740	6,4
Celkem:	**569**	**38**	**15 666**	**107,8**

Tabulka 7: Statistika výkonnostních testů pro 200 uživatelů s periodou 5 sekund

Před uložením dat do cache Redis				
	Čas [ms]			
Test	AVG	Min	Max	**Propustnost [op/min]**
CompareProducts	54 186	383	142 972	11,3
GetCategoriesByID	32 023	40	79 473	16,9
GetCategoriesHomepage	225 675	39	697 966	12,6
GetDiscussion	31 165	155	117 254	17,3
GetFilterParametersByCategory	223 527	52 989	1 260 200	9,5
GetFirmByID	15 629	37	87 114	19,3
GetProductByEAN	14 976	38	148 236	10,9
GetProductByID	34 971	352	123 673	10,6
GetProductsAll	9 848	676	124 341	12,0
GetProductsByCategory	46 799	798	156 599	10,8
GetProductsByFulltext	5 710	197	39 864	12,6
GetProductsHomepage	37 501	462	111 876	11,7
GetRequestByID	72 484	415	389 845	10,4
GetRequests	21 984	393	53 162	17,7
GetResponseByID	61 434	440	298 369	9,9
GetShopByID	12 067	38	57 042	18,7
GetUserByID	56 837	39	175 514	18,0
Celkem:	**56 283**	**37**	**1 260 200**	**230,2**

Po uložení dat do cache Redis				
	Čas [ms]			
Test	AVG	Min	Max	**Propustnost [op/min]**
CompareProducts	519	111	7 424	24,7
GetCategoriesByID	188	38	5 418	26,1
GetCategoriesHomepage	232	39	3 091	26,1
GetDiscussion	102 062	153	388 113	26,1
GetFilterParametersByCategory	780	46	17 106	11,7
GetFirmByID	116	38	1 244	35,2
GetProductByEAN	252	39	2 777	24,7
GetProductByID	104	39	1 410	27,8
GetProductsAll	4 563	701	15 097	26,0
GetProductsByCategory	78 303	12 363	434 291	24,0
GetProductsByFulltext	962	195	7 815	26,1
GetProductsHomepage	48 547	1 239	110 185	24,6
GetRequestByID	118 756	24 318	398 459	24,7
GetRequests	46 622	309	256 802	29,8
GetResponseByID	123 461	58 786	270 792	44,2
GetShopByID	124	37	978	35,2
GetUserByID	121	38	830	35,2
Celkem:	**30 924**	**37**	**434 291**	**472,2**

7 Zhodnocení budování vlastního řešení

Poslední část diplomové práce si klade za cíl zhodnotit, zda se pro portál Gloffer vyplatí budovat vlastní řešení datového rozhraní nebo využít řešení třetích stran.

Navržené a naimplementované datové rozhraní je specifické tím, že má vlastní logiku práce s daty. Samotná logika zpracování dat nemůže být nahrazena řešením třetí strany. Dá se však využít již implementovaných knihoven pro přístup k datovým zdrojům, práci s JSON formátem nebo pro hashování klíčů v Redisu. Spring Framework nám dále umožňuje pracovat s REST požadavky a vytvářením webové aplikace.

Pro vytváření datových rozhraní využívajících jazyka SQL lze použít Dibi, která poskytuje společnou logiku pro přístup k databázím jako je MySQL, MS SQL, Oracle atd. Pro využití nad NoSQL datovými zdroji však tato abstraktní knihovna není použitelná. Pro náš projekt se tedy moc nehodí. Podobnou abstraktní datovou vrstvou je MDB2, která taky poskytuje společné rozhraní pro SQL databáze.

Pro využití jazyka SQL lze do projektu přidat knihovnu překládající SQL na Elastic Search dotaz (např. Elasticsearch-SQL). Stejnou funkcionalitu poskytují knihovny pro MongoDB (např. NoSQLBooster) i Redis (např. RediSQL). Tyto knihovny však nejsou oficiálními překladači SQL syntaxe na Redis, Elastic Search nebo MongoDB dotaz. Nelze tedy spoléhat na jejich 100% funkčnost a budoucí podporu. Právě z těchto důvodů nebyly tyto knihovny pro vývoj našeho datového rozhraní využity.

8 Závěr

Cílem této diplomové práce bylo analyzovat dostupné technologie k budování webových portálů se zaměřením na platformu Linux. Dalším úkolem bylo navrhnout a implementovat nástroje pro sjednocení datových zdrojů projektu Gloffer.

V první kapitole diplomové práce byl představen koncept portálu Gloffer. Následovala kapitola popisující dostupné technologie, které jsou vhodné k budování vlastního webového portálu na platformě Linux. Zde byly jednotlivé technologie představeny a případně srovnány mezi sebou. Součástí byl výčet vlastností, které jednotlivé technologie přinášejí.

Další část popisuje návrh rozhraní pro sjednocení datových zdrojů portálu Gloffer. Je zde nastíněna architektura rozhraní pomocí diagramu nasazení, dále pak vysvětlena logika zpracování dat, tzn. kde které data získáváme a kde se musí případně uložit nebo aktualizovat. Část je věnována taky nové legislativě GDPR, která nám ukládá povinnosti, jak zacházet s daty a jak je chránit. Součástí návrhu je kapitola věnující se konvencím ukládání dat do cache databáze Redis a vlastním vytvořeným pravidlům ukládání. Další části se věnují deduplikaci výsledků vyhledávání a párování dokumentů.

Následující kapitola je věnována samotné implementaci datového rozhraní v programovacím jazyce Java s využitím Spring frameworku. Jsou zde popsány principy přidávání závislostí do projektu, vytvořené třídy, kontroléry a služby. Předposlední kapitola je věnována testování propustnosti dotazů mnou implementovaným rozhraním. Před samotným závěrem práce popisuji, zda se vyplatí budovat vlastní řešení nebo řešení třetích stran.

Velký přínos této diplomové práce vidím především v její obtížnosti, komplexnosti a využití v reálné aplikaci. Setkal jsem se při vývoji s novými technologiemi, které prohloubily mé znalosti. Jako možné rozšíření této práce by mohl být nástroj pro převod SQL dotazů na dotazy pro Elastic Search a MongoDB. Dále pak přepsání celé logiky stávající webové aplikace do tohoto rozhraní by zajistilo znovupoužitelnost ve více projektech. Úskalí práce shledávám v objemu nových technologií, se kterými jsem se musel v rámci diplomové práce seznámit a dokázat použít při implementaci.

Literatura

[1] GORMLEY, Clinton a Zachary TONG. Elasticsearch: the definitive guide. Sebastopol, CA: O'Reilly, 2015. ISBN 1449358543.

[2] SHKLAR, Leon. a Rich. ROSEN. Web application architecture: principles, protocols and practices. 2nd ed. Hoboken, NJ: Wiley, c2009. ISBN 047051860x.

[3] SHIVAKUMAR, Shailesh Kumar. Architecting high performing, scalable and available enterprise web applications. Waltham, MA: Elsevier, 2014. ISBN 9780128022580.

[4] WEERAWARANA, Sanjiva. Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more. Upper Saddle River, NJ: Prentice Hall PTR, c2005. ISBN 0131488740.

[5] SCHWARTZ, Baron, Peter ZAITSEV a Vadim TKACHENKO. High performance MySQL. 3rd ed. Beijing: O'Reilly, 2012. ISBN 9781449314286.

[6] DAYLEY, Brad. Sams Teach Yourself NoSQL with MongoDB in 24 hours. Indianapolis, Indiana: Sams, 2015. ISBN 9780672337130.

[7] SPAGGIARI, Jean-Marc a Kevin O'DELL. Architecting HBase applications: a guidebook for successful development and design. Sebastopol, CA: O'Reilly Media, 2016. ISBN 9781491915813.

[8] BROWN, Mat. Learning Apache Cassandra. 1. Birmingham: Packt Publishing, 2015. ISBN 9781783989201.

[9] PECINOVSKÝ, Rudolf. Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky. 2., aktualiz. a rozš. vyd. Praha: Grada, 2009. Myslíme v–. ISBN 9788024726533.

[10] YAMAUCHI, Owen. Hack and HHVM: Programming Productivity Without Breaking Things. USA: O'Reilly Media, 2015. ISBN 9781491920879.

[11] LERDORF, Rasmus. Speeding up the Web with PHP 7. PHP Presentation System [online]. San Francisco, 2015 [cit. 2018-02-18]. Dostupné z: http://talks.php.net/fluent15

[12] DAS, Vinoo. Learning Redis: Design efficient web and business solutions with Redis. Birmingham: Packt Publishing, 2015. ISBN 9781783980123.

[13] LUCAS, Liam. Dive into Rabbit MQ. 1. Mnichov: BookRix, 2016. ISBN 9783739653181.

[14] SNELL, James., Doug. TIDWELL a Pavel. KULCHENKO. Programming Web services with SOAP. Sebastopol, CA: O'Reilly & Associates, 2002. ISBN 0596000952.

[15] DEKA, Ganesh Chandra. NoSQL: Database for Storage and Retrieval of Data in Cloud. 1. Boca Raton: CRC Press, 2017. ISBN 9781351651622.

[16] What is MongoDB?: MongoDB [online]. 2018 [2018-04-08]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[17] LuceneImplementations: Lucene-java Wiki [online]. 2012 [cit. 2017-12-04]. Dostupné z: https://wiki.apache.org/lucene-java/LuceneImplementation

[18] Solr Features: Solr [online]. 2017 [cit. 2017-12-04]. Dostupné z: https://lucene.apache.org/solr/features.html

[19] Apache Solr: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-12-04]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Solr

[20] Základní příručka k GDPR: Úřad pro ochranu osobních údajů [online]. 2013 [cit. 2018-02-06]. Dostupné z: https://www.uoou.cz/zakladni-prirucka-k-gdpr/ds-4744/archiv=0&p1=3938

[21] Elasticsearch 6.2.0 released: Elastic [online]. 2018 [cit. 2018-02-07]. Dostupné z: https://www.elastic.co/blog/elasticsearch-6-2-0-released

[22] Moving fast with high performance Hack and PHP: HHVM [online]. 2011 [cit. 2018-02-11]. Dostupné z: https://hhvm.com/

[23] Apache Tomcat [online]. 1999 [cit. 2018-02-07]. Dostupné z: http://tomcat.apache.org/

[24] What is Memcached?: Memcached [online]. 2009 [cit. 2018-02-11]. Dostupné z: https://memcached.org/

[25] Introduction to Redis: Redis [online]. 2017 [cit. 2018-02-14]. Dostupné z: https://redis.io/topics/introduction

[26] Cloud Bigtable: Google Cloud Platform [online]. 2018 [cit. 2018-02-14]. Dostupné z: https://cloud.google.com/bigtable/

[27] Documentation: Nette [online]. 2017 [cit. 2018-02-14]. Dostupné z: https://doc.nette.org/cs/2.4

[28] Pro spring 5: an in-depth guide to the spring framework and its tools. New York, NY: Springer Science+Business Media, 2017. ISBN 9781484228074.

[29] Spring Framework Overview [online]. 2018 [cit. 2018-03-06]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/overview.html

[30] Understanding POJOs [online]. 2018 [cit. 2018-04-06]. Dostupné z: https://spring.io/understanding/POJO

[31] Cloud Bigtable architecture: Google Cloud Platform [online]. 2018 [cit. 2018-02-14]. Dostupné z: https://cloud.google.com/bigtable/img/bigtable-architecture.svg

[32] What Is Apache Hadoop?: Apache Hadoop [online]. 2017 [cit. 2018-03-03]. Dostupné z: http://hadoop.apache.org/

[33] Powered by Apache Hadoop: Hadoop Wiki [online]. 2017 [cit. 2018-03-03]. Dostupné z: https://wiki.apache.org/hadoop/PoweredBy

[34] The Underlying Technology of Messages: Facebook [online]. 2010 [cit. 2018-03-03]. Dostupné z: https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919

[35] Apache HBase ™ Reference Guide: Apache HBase [online]. 2018 [cit. 2018-04-07]. Dostupné z: https://hbase.apache.org/book.html

[36] PHP Manual: PHP [online]. 2018 [cit. 2018-02-14]. Dostupné z: http://php.net/manual/en/

[37] What is Cassandra?: Apache Cassandra [online]. 2018 [cit. 2018-03-04]. Dostupné z: http://cassandra.apache.org/

[38] The Java Programming Language Platforms: An Introduction to the Java EE Platform [online]. 2010 [cit. 2018-03-04]. Dostupné z: https://docs.oracle.com/cd/E19798-01/821-1770/gcrkk/index.html

[39] Java: Oracle Technology Network [online]. 2018 [cit. 2018-03-28]. Dostupné z: http://www.oracle.com/technetwork/java/index.html

[40] JDK 10: OpenJDK [online]. 2018 [cit. 2018-04-05]. Dostupné z: http://openjdk.java.net/projects/jdk/10/

[41] xxHash [online]. 2017 [cit. 2018-03-06]. Dostupné z: cyan4973.github.io/xxHash/

[42] Apache HTTP Server: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2018-03-02]. Dostupné z: https://en.wikipedia.org/wiki/Apache_HTTP_Server

[43] Overview of new features in Apache HTTP Server 2.4: Apache HTTP Server Project [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://httpd.apache.org/docs/trunk/new_features_2_4.html

[44] Usage of server-side programming languages for websites: W3Techs [online]. 2009 [cit. 2018-03-02]. Dostupné z: https://w3techs.com/technologies/overview/programming_language/all

[45] Zend Engine Explained: Zend Engine - In Depth [online]. 2015 [cit. 2018-03-18]. Dostupné z: http://www.zend.com/products/zend_engine/in_depth

[46] New features: Migrating from PHP 5.6.x to PHP 7.0.x [online]. 2018 [cit. 2018-04-14]. Dostupné z: http://php.net/manual/en/migration70.new-features.php

[47] Deprecated features in PHP 7.0.x: Migrating from PHP 5.6.x to PHP 7.0.x [online]. 2018 [cit. 2018-04-14]. Dostupné z: http://php.net/manual/en/migration70.deprecated.php

A Zdrojové kódy a ukázky JSON dokumentů

```
?sort=price&typeSort=desc&46247=2191916&46246=2191980&priceFrom=50&priceTo=5000
```

Výpis 14: Ukázka předávání parametrů pro účely parametrického vyhledávání a třídění


```
{
    "rating": {
        "stars_count": 4,
        "stars": 4.25,
        "unsold_count": 0,
        "percent": 0.55,
        "sold_count": 4,
        "notset_count": 8
    },
    "contacts": [
        {
            "social_linkedin": "",
            "social_instagram": "",
            "web": "",
            "address_region": null,
            "bank_name": "",
            "id": 257,
            "fax": null,
            "bank_iban": "",
            "email": "xxxxxx@xxxxxx.cz",
            "bank_account": "",
            "social_pinterest": "",
            "address_street_number": "75/12",
            "address_postal_code": "747 28",
            "address_country": "cz",
            "mobile": "",
            "social_google": "",
            "active": "enabled",
            "video_youtube": "",
            "social_twitter": "",
            "phone": "",
            "social_facebook": "",
            "bank_bic_swift": "",
            "address_street": "Zahumni",
```

```
        "address_locality": "Praha",
        "video_vimeo": ""
    }
],
"basic": {
    "firstname": "Jakub",
    "surname": "Malcharek",
    "active": "enabled",
    "login": "mal0098",
    "email": "xxxxxx@xxxxxx.cz"
},
"roles": {
    "87": {
        "role": "manager",
        "fir_firm_id": 210,
        "shp_shop_id": 87,
        "label": "Testovaci e-shop",
        "type": "shp_shop",
        "fir_branch_id": 0
    },
    "169": {
        "role": "manager",
        "fir_firm_id": 12,
        "shp_shop_id": 0,
        "label": "Vymyslena firma s.r.o.",
        "type": "fir_firm",
        "fir_branch_id": 0
    },
    "746": {
        "role": "admin",
        "fir_firm_id": null,
        "shp_shop_id": null,
        "label": "Jakub Malcharek",
        "type": "cor_user",
        "fir_branch_id": null
    }
},
"favorite": {
    "pro_catalog": {
```

```
            "72040234": "like",
            "76369901": "dislike",
            "76369930": "like",
            "76370041": "dislike",
            "76370088": "like",
            "96289272": "like"
        }
    },
    "uploaded": {
        "images": 7,
        "tickets": 2,
        "files": 0
    }
}
```


Výpis 15: Ukázka profilových informací o uživateli


```
{
    "rating": {
        "stars_count": 2,
        "stars": 4,
        "unsold_count": 0,
        "percent": 1,
        "sold_count": 2,
        "notset_count": 5
    },
    "contacts": [
        {
            "social_linkedin": "",
            "social_instagram": "",
            "web": "http://xxxxxx.cz/",
            "address_region": null,
            "bank_name": "",
            "id": 145,
            "fax": null,
            "bank_iban": "",
            "email": "xxxxxx@xxxxxx.cz",
            "bank_account": "",
            "social_pinterest": "",
            "address_street_number": "47/1",
```

```
            "address_postal_code": "708 00",
            "address_country": "cz",
            "mobile": "",
            "social_google": "",
            "active": "enabled",
            "video_youtube": "",
            "social_twitter": "",
            "phone": "",
            "social_facebook": "",
            "bank_bic_swift": "",
            "address_street": "Lichnovskeho",
            "address_locality": "Ostrava",
            "video_vimeo": ""
        }
    ],
    "basic": {
        "firm": 224,
        "icon": null,
        "description": "testtesttesttesttest",
        "active": "enabled",
        "label": "testtest",
        "title": "testtest",
        "url": "http://xxxxxx.cz/",
        "email": "xxxxxx@xxxxxx.cz",
        "content": null
    }
}
```


Výpis 16: Ukázka profilových informací o e-shopu


```
{
    "rating": {
        "stars_count": 6,
        "stars": 3.67,
        "unsold_count": 0,
        "percent": 0.93,
        "sold_count": 6,
        "notset_count": 13
    },
    "contacts": [
```

```
    {
        "social_linkedin": "",
        "social_instagram": "",
        "web": "",
        "address_region": "Moravskoslezsky kraj",
        "bank_name": "",
        "id": 256,
        "fax": "",
        "bank_iban": "",
        "email": "",
        "bank_account": "",
        "social_pinterest": "",
        "address_street_number": "87/4",
        "address_postal_code": "724 00",
        "address_country": "cz",
        "mobile": "",
        "social_google": "",
        "active": "disabled",
        "video_youtube": "",
        "social_twitter": "",
        "phone": "",
        "social_facebook": "",
        "bank_bic_swift": "",
        "address_street": "17. listopadu",
        "address_locality": "Ostrava",
        "video_vimeo": ""
    }
],
"basic": {
    "in": "xxxxxxxxxxxxx",
    "phone": "xxxxxxxxx",
    "icon": null,
    "vat": "xxxxxxxxxxxxxx",
    "description": "testtesttesttesttesttest",
    "active": "enabled",
    "label": "Testovaci firma",
    "title": "Testovaci firma",
    "url": null,
    "email": "xxxxxx@xxxxxx.cz",
```

```
        "content": null
    },
    "shops": [
        {
            "id": 87,
            "title": "Testovaci e-shop"
        },
        {
            "id": 88,
            "title": "Test 2"
        }
    ]
}
```


Výpis 17: Ukázka profilových informací o firmě


```
{
   "keys": [
      "api:product:ean:hint:4008496892402"
   ],
   "domains": [
      "api:product",
      "api:products"
   ]
}
```


Výpis 18: Ukázka těla REST požadavku pro aktualizaci cache databáze Redis


```
{
   "email": "mal0098@vsb.cz",
   "password": "xxxxxxxxx",
   "firstname": "Jakub",
   "surname": "Malcharek"
}
```


Výpis 19: Ukázka těla REST požadavku pro registraci uživatele

```
{
   "query": {
      "database": "develop",
      "find": {
         "$or" : [
            {
               "mapping.pro_catalog_id" : 39142552
            },
            {
               "mapping.pro_catalog_id" : 50415510
            }
         ]
      }
   }
}
```

Výpis 20: Ukázka těla REST požadavku pro dotazy nad MongoDB

```
{
   "query": "SELECT * FROM test3.testuji"
}
```

Výpis 21: Ukázka těla REST požadavku pro DML operace MySQL databáze

```
{
   "query": {
      "type": "select",
      "key": "user:601"
   }
}
```

Výpis 22: Ukázka těla REST požadavku pro dotazy nad Redis databází

```
{
   "query": {
      "method": "GET",
      "endpoint": "gloffer_data/pro_catalog/_search",
      "command": {
         "_source": "id",
         "query" : {
            "ids" : {
               "type" : "pro_catalog",
               "values" : [
                  "63434515"
               ]
            }
         }
      }
   }
}
```

Výpis 23: Ukázka těla REST požadavku pro dotazy nad Elastic Search databází

```
{
   "user_id": 746,
   "object_type": "pro_catalog",
   "favorite_type": "dislike",
   "active": "enabled"
}
```

Výpis 24: Ukázka těla REST požadavku pro nastavení oblíbenosti produktu

```
{
  "id": 68135281,
  "title": {
    "cs": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/MC360
       (15k) - originalni",
    "sk": null,
    "en": null
  }
}
```

Výpis 25: Ukázka informací o produktu ve verzi *hint*

```
{
  "id": 68135281,
  "title": {
    "cs": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/MC360
       (15k) - originalni",
    "sk": null,
    "en": null
  },
  "price": null,
  "currency": "czk",
  "source": "pro_catalog",
  "image": []
}
```


Výpis 26: Ukázka informací o produktu ve verzi *simple*


```
{
  "id": 68135281,
  "title": {
    "cs": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/MC360
       (15k) - originalni",
    "sk": null,
    "en": null
  },
  "price": null,
  "currency": "czk",
  "source": "pro_catalog",
  "image": [],
  "description": {
    "cs": "Vyrobce: OKI\n - GlofferDesc",
    "sk": null,
    "en": null
  },
  "property": [
    {
      "category_property_group_id": null,
      "code": "param-name-15066",
      "icon": null,
      "show": "checkboxlist",
```

```
    "type": "string",
    "show_new": "checkboxlist",
    "category_property_group_multiple_order": 0,
    "property_id": 39332,
    "value_single_id": null,
    "multi": [
     {
       "value_icon": null,
       "value_type": "value",
       "value_title": "OKI",
       "property_value_parent_id": 1317336,
       "value_order": 1000,
       "language": "cs",
       "value_content": null,
       "property_id": 39332,
       "value_label": "oki",
       "value_code": "39332|oki|",
       "value_translating": "no",
       "id": 61935163,
       "value_number": null,
       "value_description": "OKI",
       "value": "39332|oki|",
       "unit_id": null
     }
    ],
    "dependency_id": null,
    "property_multiple_order": 0,
    "unit_default_id": null,
    "value_list_tag": "no",
    "unit_code": null,
    "id": 91842753,
    "value": null,
    "unit_id": null,
    "slug": "2904673_Vyrobce"
  }
],
"product_feed": [
  {
    "images": [
```

```
        "http://www.glofik.cz/File/Fotocache/Feed/_185/185227_0a.jpg?ZnJvc3Q="
      ],
      "product_url": "http://www.glofik.cz/cz/produkt/oki-obraz-valec-pro-toner
         -cyan-do-c3520-mfp-c3530-mfp-mc350-mc360-15k",
      "ean": "5031713037866",
      "fir_firm_id": 169,
      "price": null,
      "shp_shop_id": 45,
      "name": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/
         MC360 (15k)\n          - Oki",
      "vat": null
    },
    {
      "images": [
        "http://www.glofik.cz/File/Fotocache/Feed/_293/293272_0a.jpg?ZnJvc3Q="
      ],
      "product_url": "http://www.glofik.cz/cz/produkt/oki-toner-cyan-do-c3520-
         mfp-c3530-mfp-mc350-mc360-2-5k",
      "ean": "5031713045434",
      "fir_firm_id": 169,
      "price": null,
      "shp_shop_id": 45,
      "name": "Oki Toner Cyan do C3520 MFP/C3530 MFP/MC350/MC360 (2.5k)\n
                    - Oki",
      "vat": null
    }
  ],
  "category_id": 42724,
  "trademark": {
    "id": null,
    "title": null
  }
}
```


Výpis 27: Ukázka informací o produktu ve verzi *full*

```
{
  "id": 68135281,
  "title": {
    "cs": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/MC360
       (15k) - originalni",
    "sk": null,
    "en": null
  },
  "price": null,
  "currency": "czk",
  "source": "pro_catalog",
  "image": [],
  "description": {
    "cs": "Vyrobce: OKI\n - GlofferDesc",
    "sk": null,
    "en": null
  },
  "property": [
    {
      "category_property_group_id": null,
      "code": "param-name-15066",
      "icon": null,
      "show": "checkboxlist",
      "type": "string",
      "show_new": "checkboxlist",
      "category_property_group_multiple_order": 0,
      "property_id": 39332,
      "value_single_id": null,
      "multi": [
        {
          "value_icon": null,
          "value_type": "value",
          "value_title": "OKI",
          "property_value_parent_id": 1317336,
          "value_order": 1000,
          "language": "cs",
          "value_content": null,
          "property_id": 39332,
```

```
        "value_label": "oki",
        "value_code": "39332|oki|",
        "value_translating": "no",
        "id": 61935163,
        "value_number": null,
        "value_description": "OKI",
        "value": "39332|oki|",
        "unit_id": null
      }
    ],
    "dependency_id": null,
    "property_multiple_order": 0,
    "unit_default_id": null,
    "value_list_tag": "no",
    "unit_code": null,
    "id": 91842753,
    "value": null,
    "unit_id": null,
    "slug": "2904673_Vyrobce"
  }
],
"product_feed": [
  {
    "images": [
      "http://www.glofik.cz/File/Fotocache/Feed/_185/185227_0a.jpg?ZnJvc3Q="
    ],
    "product_url": "http://www.glofik.cz/cz/produkt/oki-obraz-valec-pro-toner
       -cyan-do-c3520-mfp-c3530-mfp-mc350-mc360-15k",
    "ean": "5031713037866",
    "fir_firm_id": 169,
    "price": null,
    "shp_shop_id": 45,
    "name": "Oki Obraz. valec pro toner cyan do C3520 MFP/C3530 MFP/MC350/
       MC360 (15k)\n          - Oki",
    "vat": null
  }
],
"category_id": 42724,
"trademark": {
```

```
    "id": null,
    "title": null
  },
  "content": "<p>zobrazovaci valec pro azurovy toner, az 15tis. str. pri 5%
     pokryti, pro C3520 MFP,C3530 MFP, MC350, MC360</p>",
  "ean": null,
  "stats": {
    "inspection_year": 0,
    "inspection_month": 0,
    "inspection_full": 0,
    "sales_count": 0,
    "favorite_dislike": 0,
    "favorite_like": 0,
    "request_count": 0
  }
}
```


Výpis 28: Ukázka informací o produktu ve verzi *huge*


```
[
  {
    "id": 238,
    "title": {
      "cs": "Chovatelstvi",
      "sk": "Chovatelstvo",
      "en": "Animals & Pet Supplies"
    }
  },
  {
    "id": 359,
    "title": {
      "cs": "Obleceni a doplnky",
      "sk": "Oblecenie a doplnky",
      "en": "Apparel & Accessories"
    }
  },
   ...
]
```


Výpis 29: Ukázka informací o potomcích kategorií

```
{
   "42145": {
      "values": [
         {
            "1704897": "Boma"
         },
         {
            "1704900": "Force"
         },
         {
            "1704902": "Author"
         }
      ]
   },
   ...
}
```

Výpis 30: Ukázka vlastností kategorie pro možnosti filtrování

```
{
  "path": "/products/233/hint/ScreeShield/0",
  "parameters": {
    "sort": [
      "price"
    ],
    "typeSort": [
      "asc"
    ]
  }
}
```

Výpis 31: Ukázka informací uložených v reference

```
{
   "user_id": 746,
   "firm_id": 210,
   "product_id": 68135281,
   "amount": 3
}
```

Výpis 32: Ukázka těla REST požadavku pro vytvoření poptávky

```
{
   "user_id": 593,
   "shop_id": 45,
   "request_id": 204,
   "price": 50
}
```

Výpis 33: Ukázka těla REST požadavku pro vytvoření nabídky

```
{
   "user_id": 746,
   "request_id": 53,
   "buyer_type": "cor_user",
   "buyer_id": 746
}
```

Výpis 34: Ukázka těla REST požadavku pro vytvoření diskuse

```
{
   "price": 50
}
```

Výpis 35: Ukázka těla REST požadavku pro aktualizaci nabídky

```
{
   "user_id": 746,
   "content": "Zkousim, zda to funguje ...",
   "message_creator": "buyer"
}
```

Výpis 36: Ukázka těla REST požadavku pro přidání zprávy do diskuse

```
{
   "conclusion": "sold",
   "rating": "excellent",
   "comment": "Nejlepsi prodavajici :)"
}
```

Výpis 37: Ukázka těla REST požadavku pro ohodnocení prodávajícího

```
for (String k : allKeys) {
   String[] parts = k.split(":");
   if (parts[0].compareTo("api") == 0) {
      if (parts[1].compareTo("reference") != 0) {
         if (parts[1].compareTo("products") == 0 && parts[2].compareTo("all") ==
              0) {
            dataService.insertToRedis(k, "{}", 0);
            String tmp = "/" + parts[1] + "/" + parts[2];
            getProductsAll(tmp);
         } else if (parts[1].compareTo("product") == 0 && parts[2].chars().
             allMatch(Character::isDigit)) {
            dataService.insertToRedis(k, "{}", 0);
            String tmp = "/" + parts[1] + "/" + parts[2] + "/" + parts[3];
            getProduct(Integer.parseInt(parts[2]), parts[3], tmp);
         } else if (parts[1].compareTo("product") == 0 && parts[2].compareTo("
             ean") == 0) {
            dataService.insertToRedis(k, "{}", 0);
            String tmp = "/" + parts[1] + "/" + parts[2] + "/" + parts[3] + "/"
                + parts[4];
            getProdByEAN(parts[4], parts[3], tmp);
         } else if (parts[1].compareTo("products") == 0 && parts[2].chars().
             allMatch(Character::isDigit) && parts.length == 6) {
            String found = dataService.findInRedis(AppConfig.
                getReferenceRedisPath() + parts[5]);
            Map<String, Object> map = null;
            try {
               map = mapper.readValue(found, HashMap.class);
            } catch (IOException e) {
               return false;
            }
            dataService.insertToRedis(k, "{}", 0);
            dataService.insertToRedis(AppConfig.getReferenceRedisPath() + parts
                [5], "{}", 0);
            getProdByCat(Integer.parseInt(parts[2]), Integer.parseInt(parts[4]),
                 parts[3], (String) map.get("path"), getParams(map));
         } else if (parts[1].compareTo("products") == 0 && parts.length == 3 &&
             parts[2].compareTo("all") != 0) {
```

```
        String found = dataService.findInRedis(AppConfig.
            getReferenceRedisPath() + parts[2]);
        Map<String, Object> map = null;
        try {
            map = mapper.readValue(found, HashMap.class);
        } catch (IOException e) {
            return false;
        }
        dataService.insertToRedis(k, "{}", 0);
        dataService.insertToRedis(AppConfig.getReferenceRedisPath() + parts
            [2], "{}", 0);
        String[] par = ((String) map.get("path")).split("/");
        getProducts(par[3], par[2], (String) map.get("path"));
    } else if (parts[1].compareTo("products") == 0 && parts[2].compareTo("
        homepage") == 0) {
        String found = dataService.findInRedis(AppConfig.
            getReferenceRedisPath() + parts[3]);
        Map<String, Object> map = null;
        try {
            map = mapper.readValue(found, HashMap.class);
        } catch (IOException e) {
            return false;
        }
        dataService.insertToRedis(k, "{}", 0);
        dataService.insertToRedis(AppConfig.getReferenceRedisPath() + parts
            [3], "{}", 0);
        getProductsHomepage((String) map.get("path"), getParams(map));
    } else if (parts[1].compareTo("search") == 0 && parts.length == 4) {
        String found = dataService.findInRedis(AppConfig.
            getReferenceRedisPath() + parts[2] + ":" + parts[3]);
        Map<String, Object> map = null;
        try {
            map = mapper.readValue(found, HashMap.class);
        } catch (IOException e) {
            return false;
        }
        dataService.insertToRedis(k, "{}", 0);
        dataService.insertToRedis(AppConfig.getReferenceRedisPath() + parts
            [2] + ":" + parts[3], "{}", 0);
```

```
            String[] par = ((String) map.get("path")).split("/");
            findFulltext(Integer.parseInt(par[2]), par[4], Integer.parseInt(par
                [5]), par[3], (String) map.get("path"), getParams(map));
        }
      }
   }
}
```

Výpis 38: Nástroj pro hromadnou aktualizaci záznamů v cache Redis

B Obsah CD

Přiložené CD obsahuje tyto adresáře:

- **App** - Adresář se zdrojovými kódy nástroje pro sjednocení datových zdrojů
- **Documentation** - Adresář s HTML dokumentací
- **Images** - Adresář s diagramy a obrázky
- **JavaDoc** - JavaDoc dokumentace vytvořeného projektu
- **JMeter** - Scénář výkonnostního testování a výsledky

Přiložené CD obsahuje tyto soubory:

- **MAL0098_DP.pdf** - Text diplomové práce

C Statistika vytvořených souborů

Tabulka 8: Statistika vytvořených souborů v rámci implementace

	Počet řádků			
Soubor	Celkově	Kód	Komentáře	Prázdné řádky
AppConfig.java	479	293	108	78
Category.java	37	25	4	8
CategoryDataService.java	20	5	13	2
CategoryDataServiceImpl.java	112	85	4	23
CommonData.java	107	93	4	10
DataRestController.java	719	481	198	40
DataService.java	66	12	45	9
DataServiceImpl.java	528	394	62	72
Discussion.java	290	229	4	57
FirmData.java	65	47	4	14
FirmDataService.java	22	5	15	2
FirmDataServiceImpl.java	194	145	16	33
Logger.java	94	67	20	7
Message.java	193	146	4	43
MySQLUtils.java	32	19	10	3
ProductDataService.java	97	15	70	12
ProductDataServiceImpl.java	1858	1392	144	322
ProductFull.java	81	58	4	19
ProductHint.java	45	32	4	9
ProductHuge.java	48	34	4	10
ProductSimple.java	74	52	4	18
RedisConfig.java	50	35	12	3
Request.java	633	487	4	142
RequestDataService.java	139	21	100	18
RequestDataServiceImpl.java	1538	1252	15	271
Response.java	653	503	4	146
ResponseData.java	87	66	4	17
ShopData.java	38	25	4	9
ShopDataService.java	22	5	15	2
ShopDataServiceImpl.java	174	131	13	30
SpringbootJwtApplication.java	28	22	0	6
Tests.java	315	295	8	12
UserData.java	87	69	4	14
UserDataService.java	37	7	26	4
UserDataServiceImpl.java	396	298	28	70
Utils.java	91	47	34	10
Celkem:	**9449**	**6892**	**1012**	**1545**